

C

UNIVERSITY OF
ILLINOIS LIBRARY
AT URBANA-CHAMPAIGN
ENGINEERING

NOTICE: Return or renew all Library Materials! The *Minimum Fee* for each Lost Book is \$50.00.

JUL 06 1988

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.
To renew call Telephone Center, 333-8400

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

L161—O-1096

10. 87
2630
20, 252
op. 1
ENGINEERING LIBRARY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS

Engin

CONFERENCE ROOM

Center for Advanced Computation

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
URBANA, ILLINOIS 61801

CAC Document Number 252
CCTC-WAD Document Number 7523

Research in
Network Data Management and
Resource Sharing

INTELLIGENT TERMINAL SOFTWARE FLOWCHARTS

October 31, 1977

The Library of the

MAY 21 1978

University of Illinois

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

ENGINEERING

JUN 6 1978

MAR 1 1982

MAR 7 1982
3 REC'D

Intelligent Terminal
Software Flowcharts

Deborah S. Brown
Betty Kasprzycki
John R. Mullen
David A. Willcox

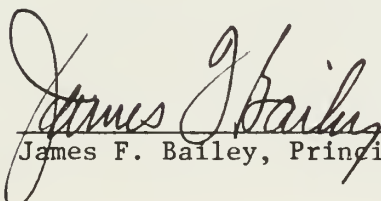
Prepared for the
Command and Control Technical Center
WWMCCS ADP Directorate
Defense Communication Agency
Washington, D.C.

under contract
DCA100-76-C-0088


Center for Advanced Computation
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

October 31, 1977

Approved for release:



James F. Bailey, Principal Investigator



Digitized by the Internet Archive
in 2012 with funding from
University of Illinois Urbana-Champaign

<http://archive.org/details/intelligenttermi252brow>

INTRODUCTION

ORGANIZATION

This document contains Nassi-Schneiderman flow charts for the software comprising the operating system and standard support package for the Intelligent Terminal operating system. Readers should refer to the Intelligent Terminal Programmer's Manual (CCTC-WAD document #7616) for a description of how these routines interact, and for descriptions of the proper usage of these routines.

The flow charts on the following pages are arranged alphabetically by routine name. In most cases there is one chart for each routine. However, some charts are too complex to be presented legibly on a single page. In each of these cases, one or more sections of the chart have been broken out and placed on a following page. If a notation such as "See ph_driver: read_type" appears in a chart, then a sub-chart labeled "ph_driver: read_type" will appear on one of the immediately following pages.

There are two implementations of the Intelligent Terminal software. One of these runs on a Digital Equipment Corporation LSI-11 minicomputer, and the other runs on Honeywell Level 6 minicomputers. Most routines are identical in the two implementations. A few routines are implemented differently on the two machines, primarily due to fundamental differences in the structure of the hardware base. Each of these routines has two charts, one for the LSI-11 version and one for the Level 6 implementation.

NASSI-SHNEIDERMAN DIAGRAMS

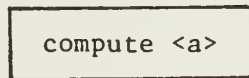
The diagramming technique developed by I. Nassi and B. Shneiderman (cf. SIGPLAN Notices, August 1973) provides four basic visual structures corresponding to the four basic constructs of a program:

1. process,
2. decision,
3. multi-case decision, and
4. iteration.

The Nassi-Shneiderman visual structures corresponding to these program constructs are described below.

Process

A process (meaning any computation) is represented by a box as follows:

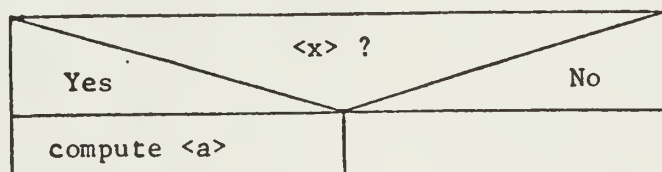


The box is usually named, or some English phrase or some equation is written in the box to indicate the nature of the process or computation. The box may represent any process or computation, from the whole of an operating system to a single statement of the kind " $a = b + c$ ". An empty box represents the null process: "do nothing".

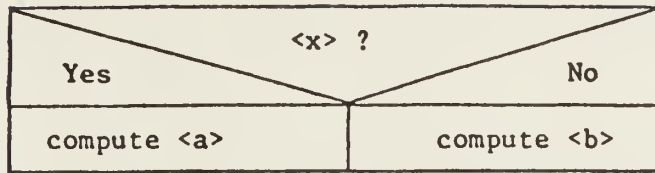
Decision

The two most common decisions are represented by the if statement and the if...else statement. These two decisions are represented as follows:

1. if statement:

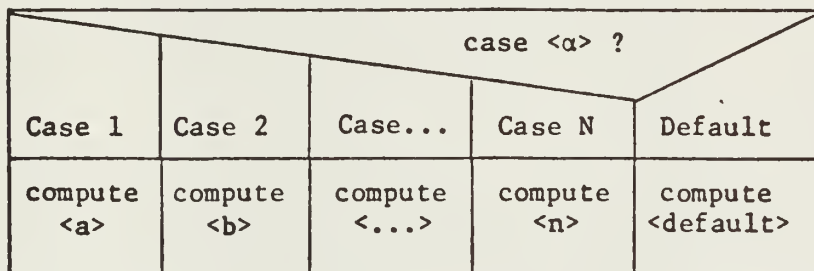


2. if ... else statement:



Multi-case Decision

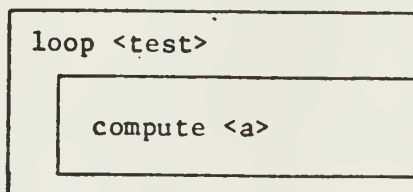
The representation of a multi-case decision is a simple extension of the previous visual structure for representing simple decisions:



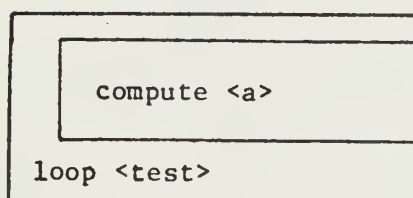
Iteration

The two most common forms of iteration are those with a top test and those with a bottom test. These two forms of iteration are represented as follows:

1. top test:

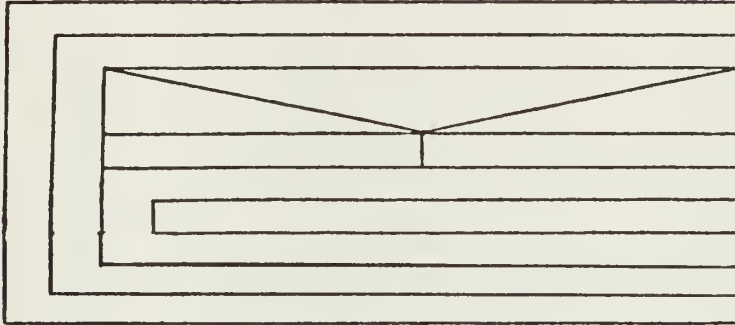


2. bottom test:



Combination

The visual structures presented above may be combined to any degree to represent a computational structure, e.g.:



alloc(size)

For every used entry in CORETAB.

Is the size of this entry
greater than or equal
to size?

NO

YES

Remember the address of this entry.

Move the beginning of the entry
to after this piece.

Decrement the size of the entry
by the size of the piece being
allocated.

Is size of entry now 0?

NO

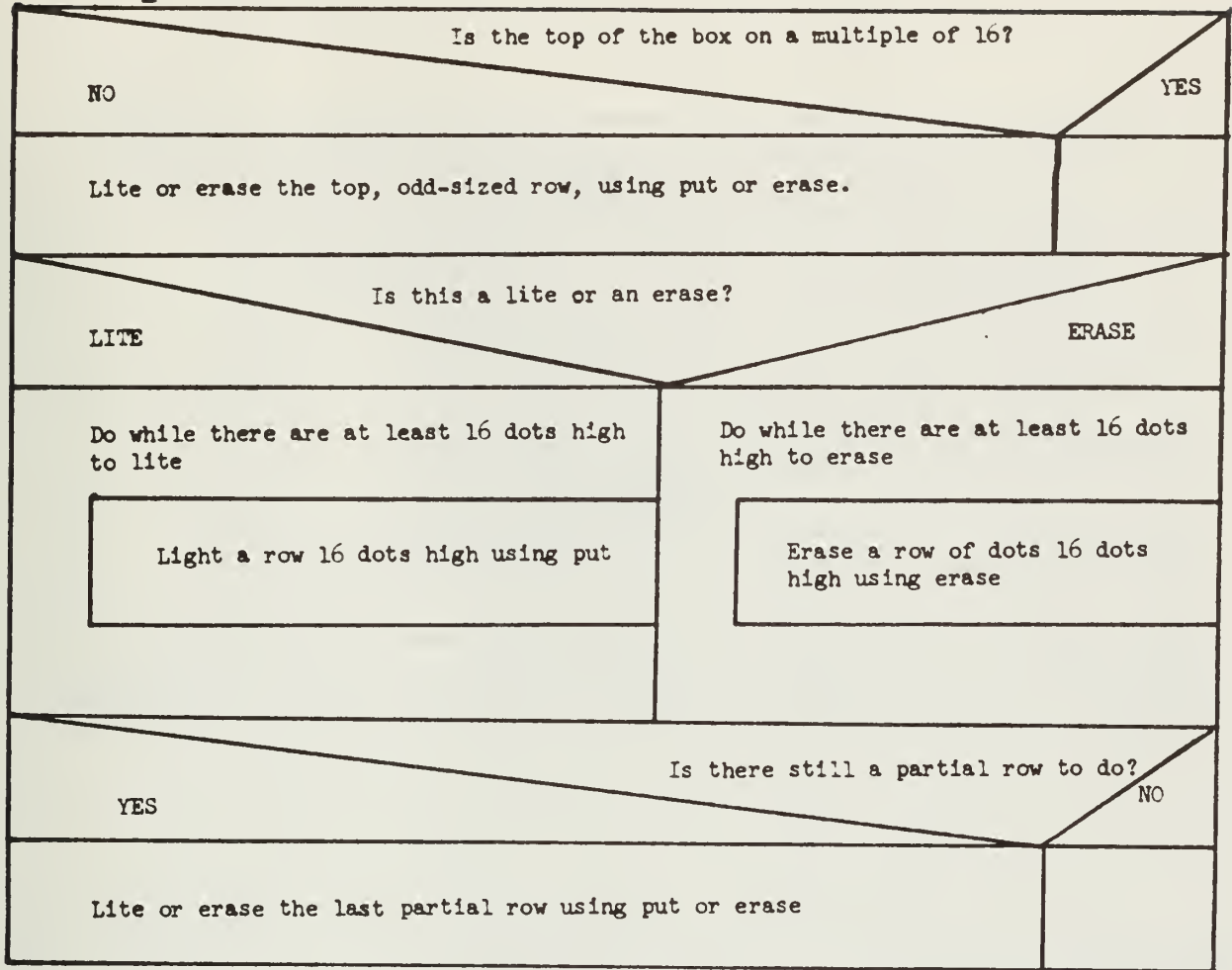
YES

Delete this empty entry by
copying rest of CORETAB up
one slot.

Return the remembered address.

Return -1.

area_lite(x1,y1,x2,y2,mode)



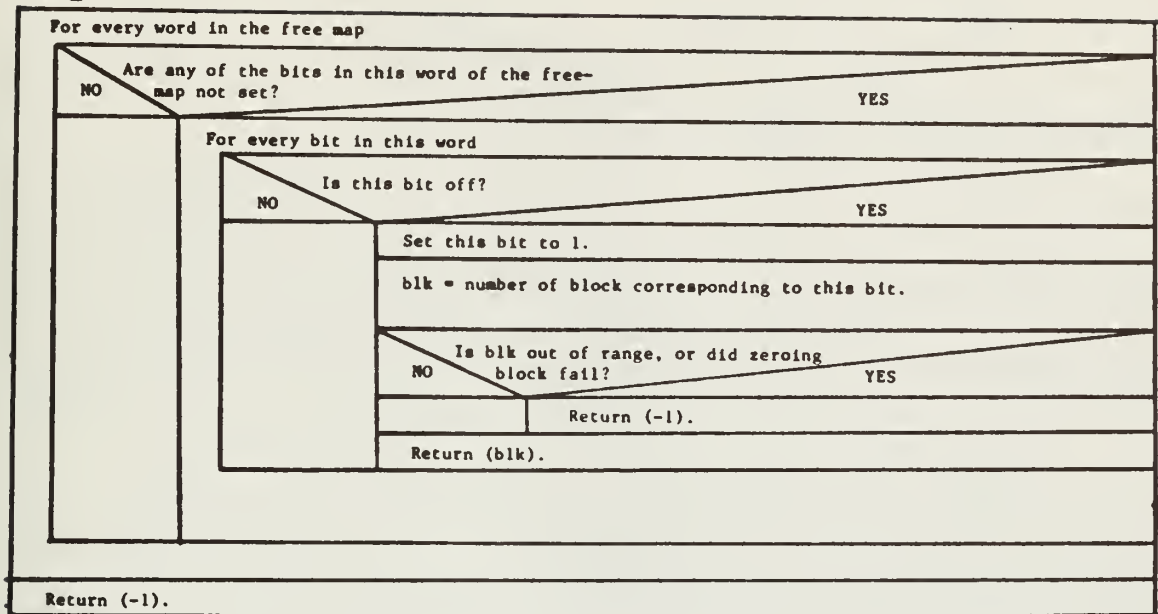
NOTE: This version of area_lite is specific to the LSI-11 IT.

area_lite

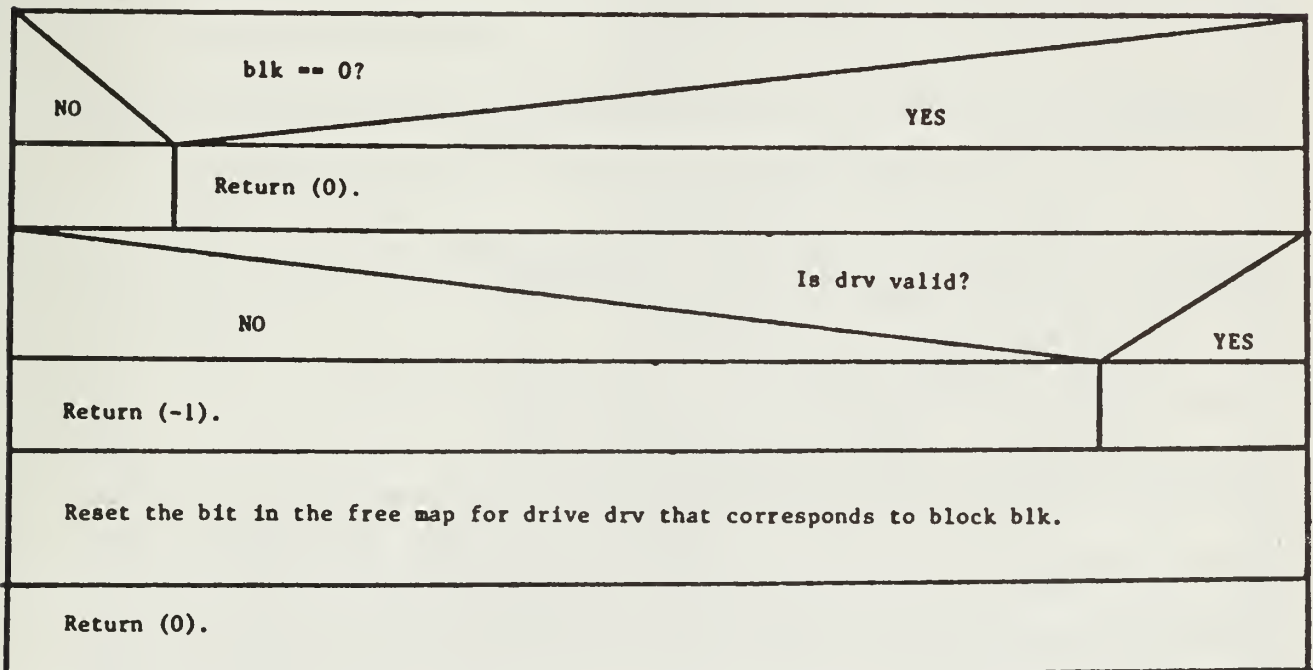
Write an appropriately formatted message to the Z80 panel controller.

NOTE: This version of area_lite is specific to the L6 IT.

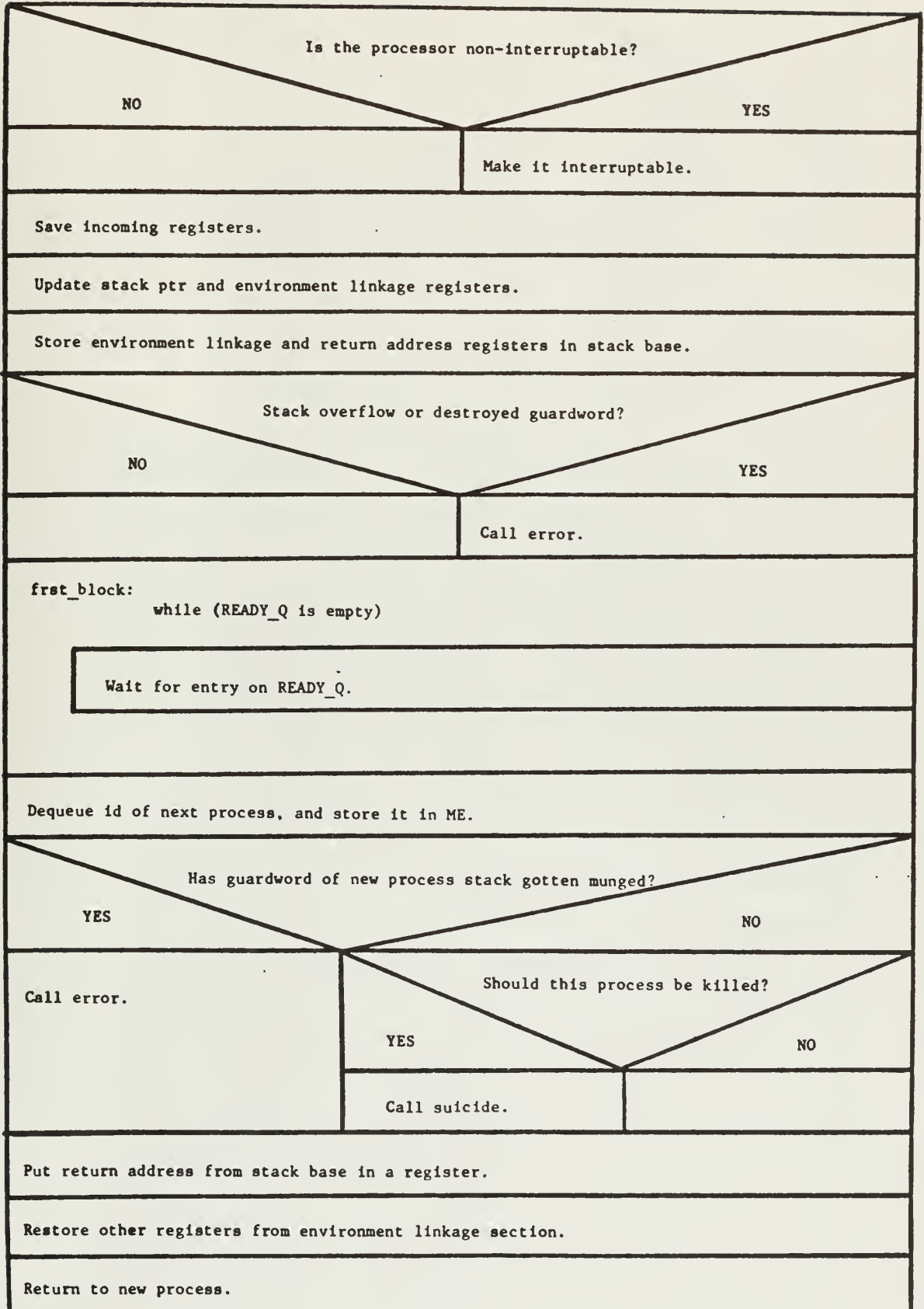
blk_alloc(drv)



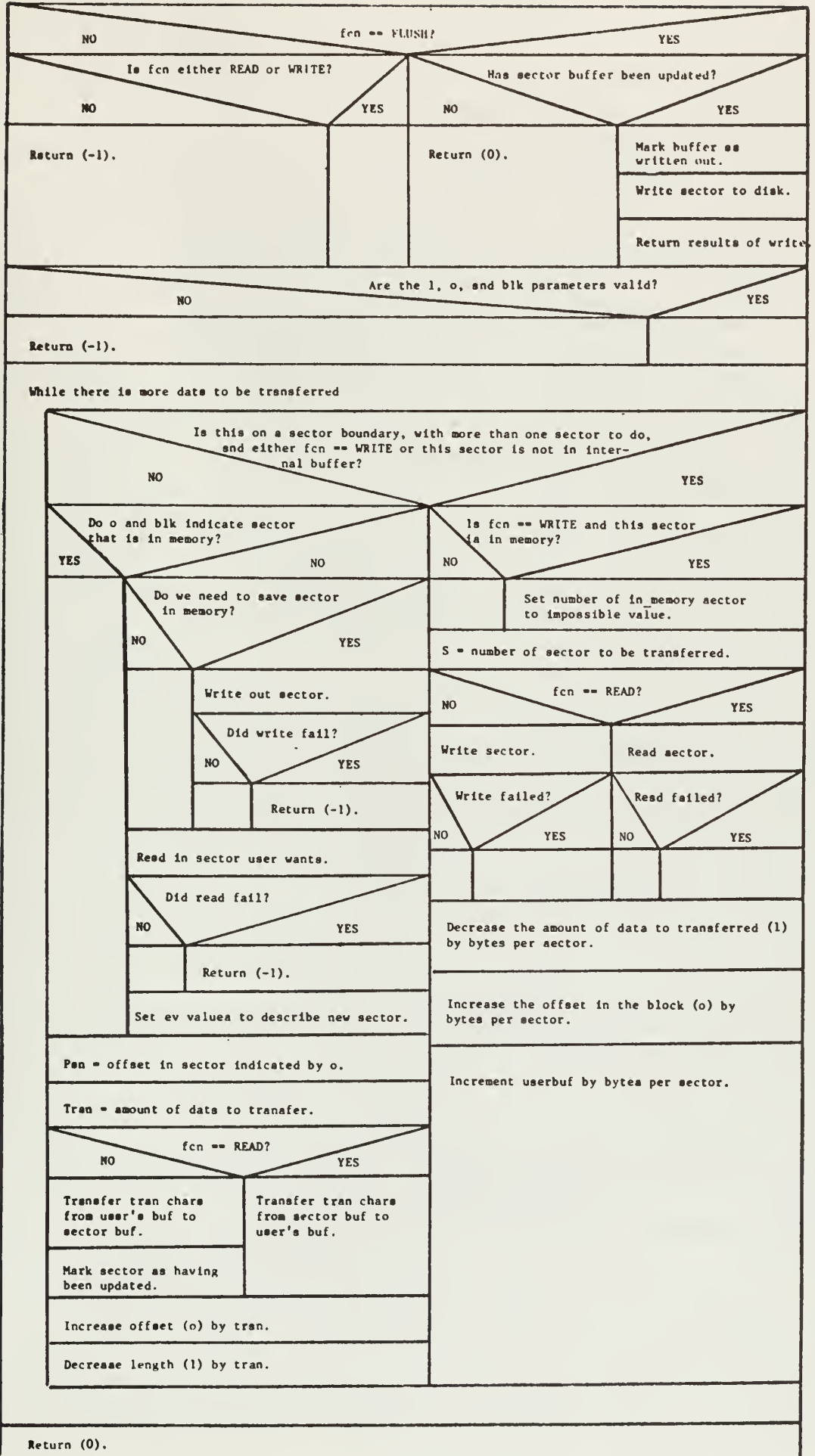
blk_free(drv, blk)



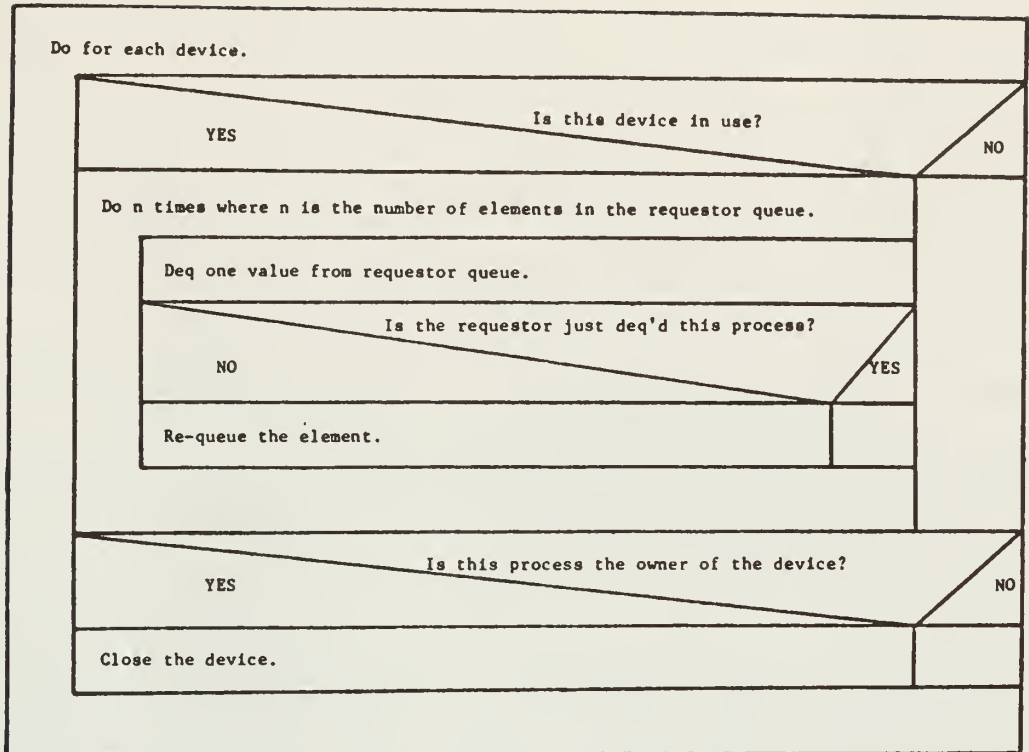
block()



buflo(ev, blk, o, userbuf, l, fcn)

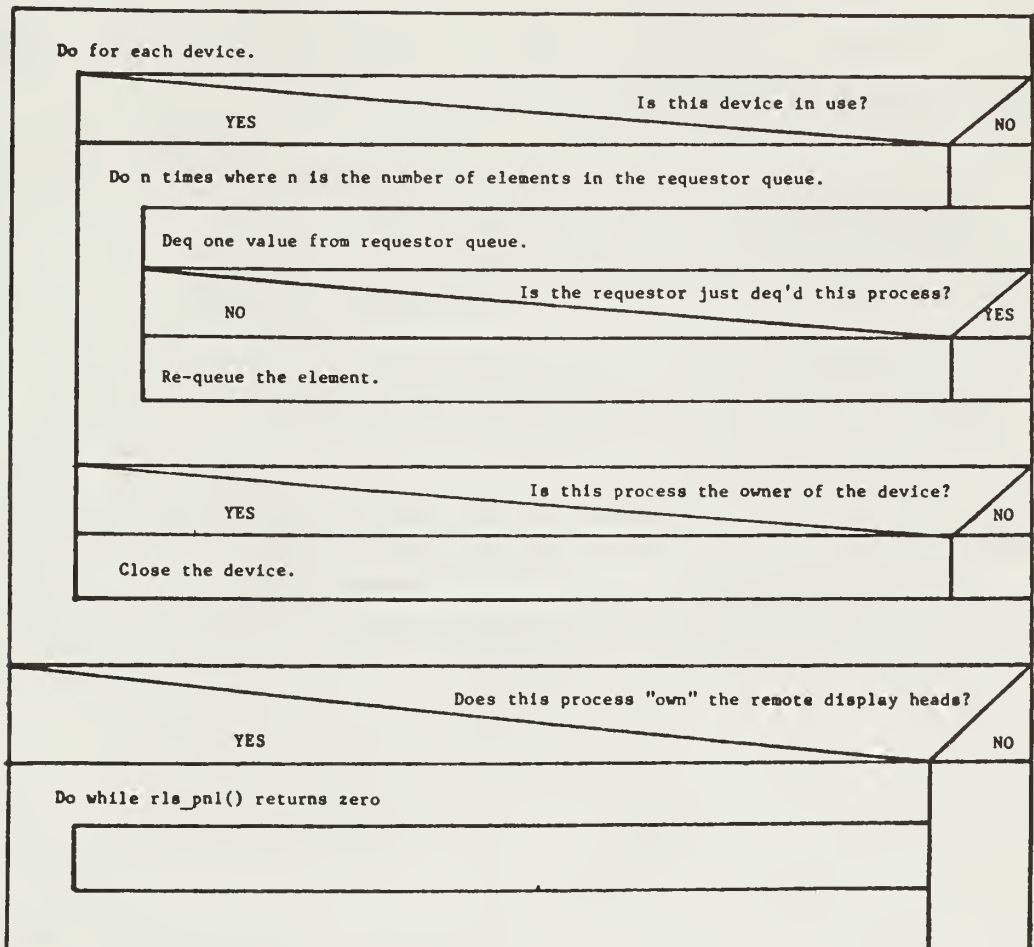


clear_io()



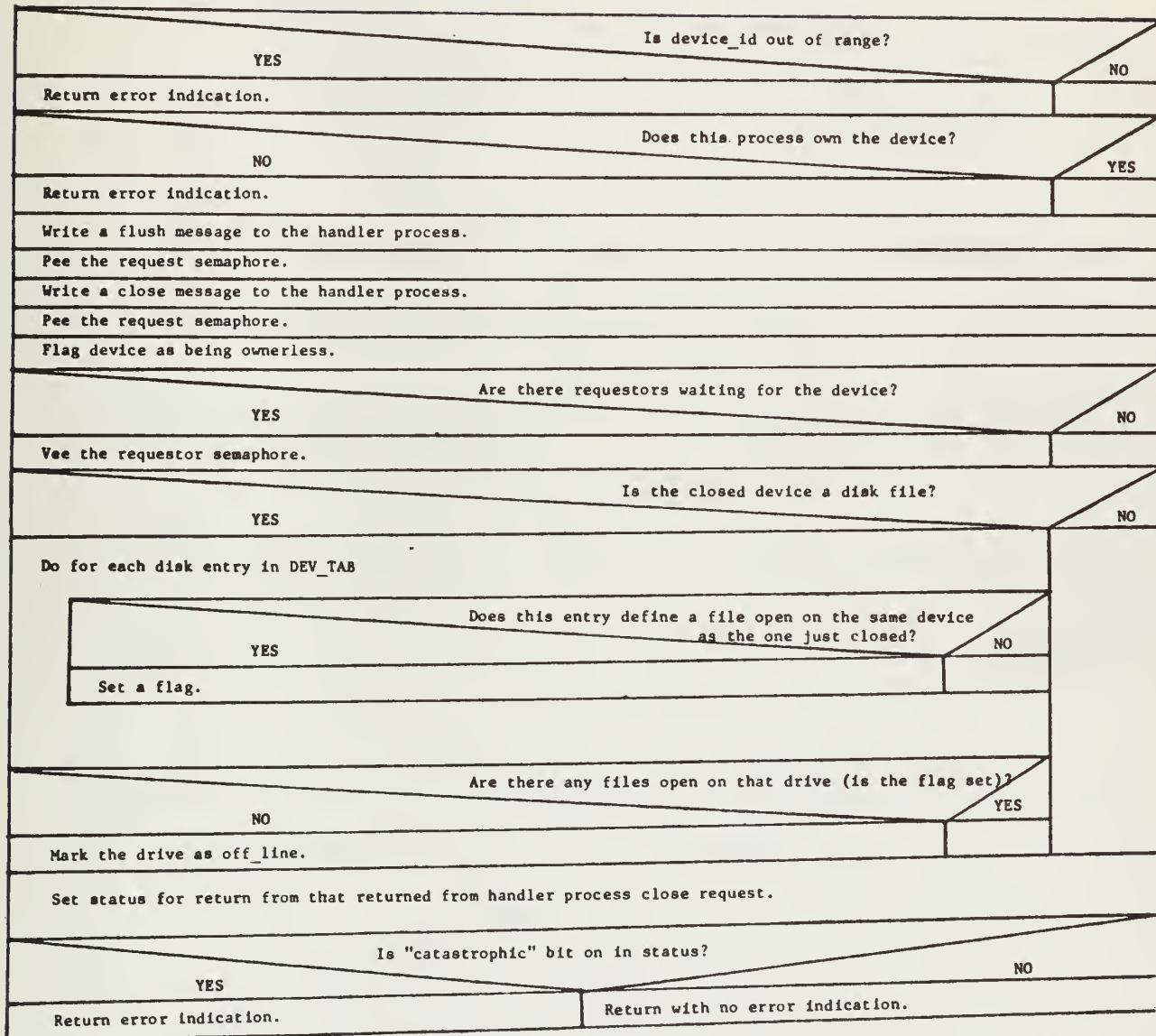
NOTE: This version of clear_io is specific to the LSI 11 IT.

clear_io()

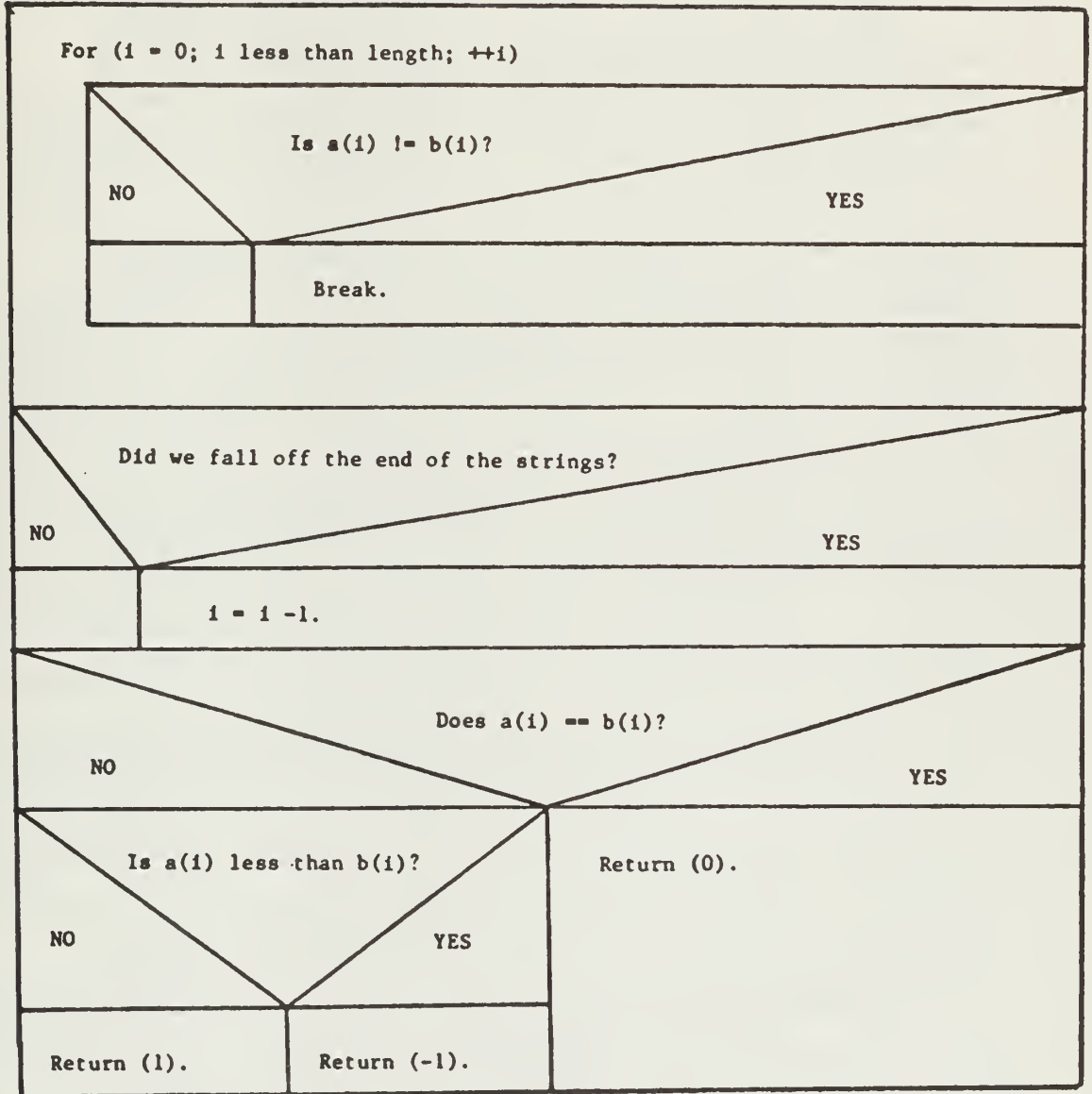


NOTE: This version of clear_io is specific to the Level 6 IT.

close(device_id, &status)

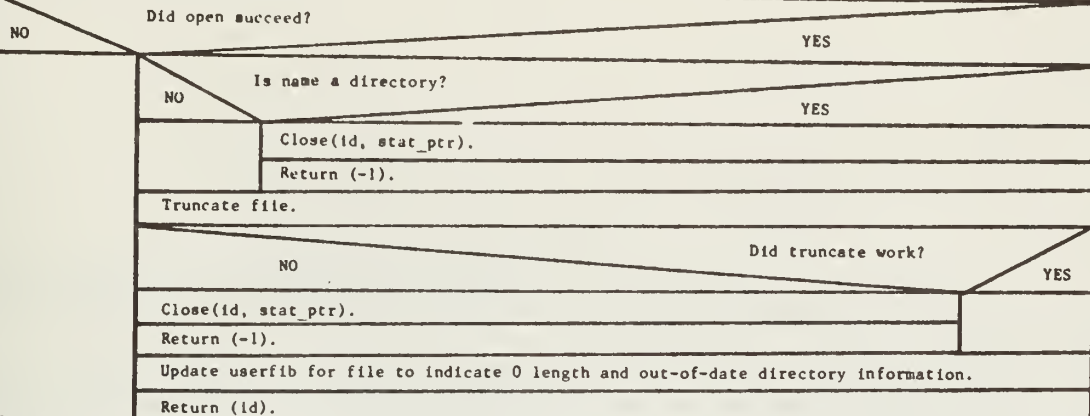


cmp (a, b, length)



create(name, stat_ptr)

Id = open(name, 0, stat_ptr)



Temporarily terminate name at end of name of containing directory.

Open the containing directory and store results in id.



Restore terminated name.

a = pointer into name after delimiter after containing directory.

off = 0.

For each entry in the directory.

Read the directory entry.



Increment off by size of directory entry.

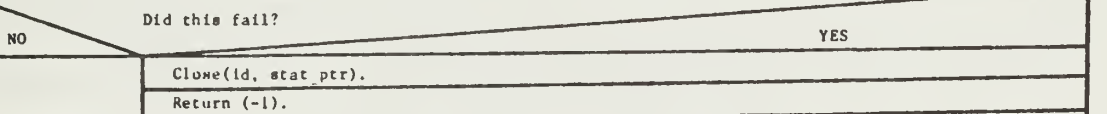
Seek in directory to beginning of empty directory.

Create a directory entry structure by copying the file name into it and setting the flags and lengths to indicate a zero length new file.

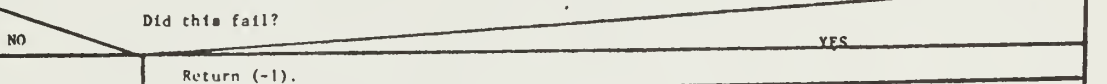
Allocate a disk block to use as index block.



Write the directory entry structure to the directory.



Close the containing directory.



Id = open(name, 0, stat_ptr).

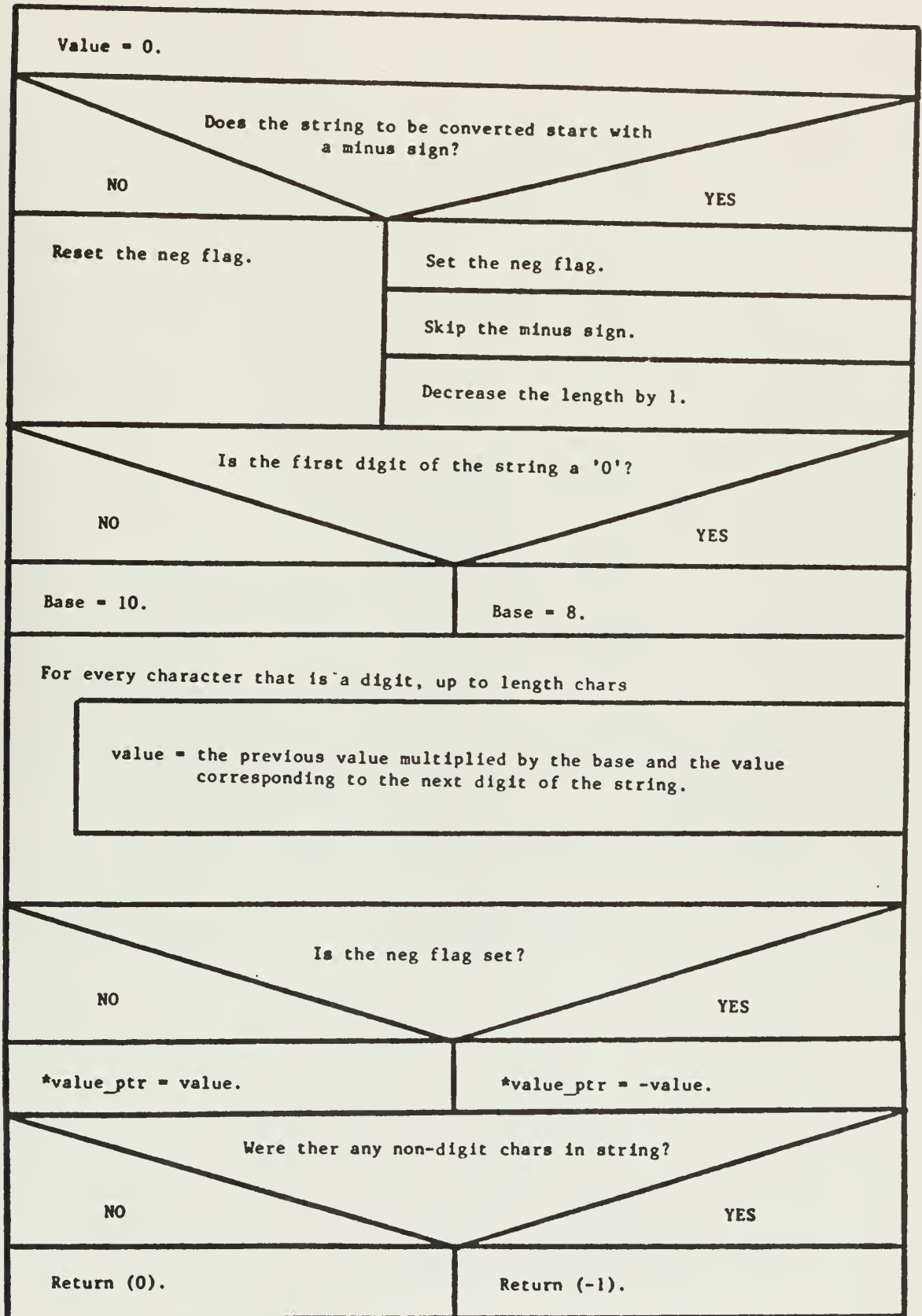
Return (id).

creep(stack_size, proc, parm, priority)

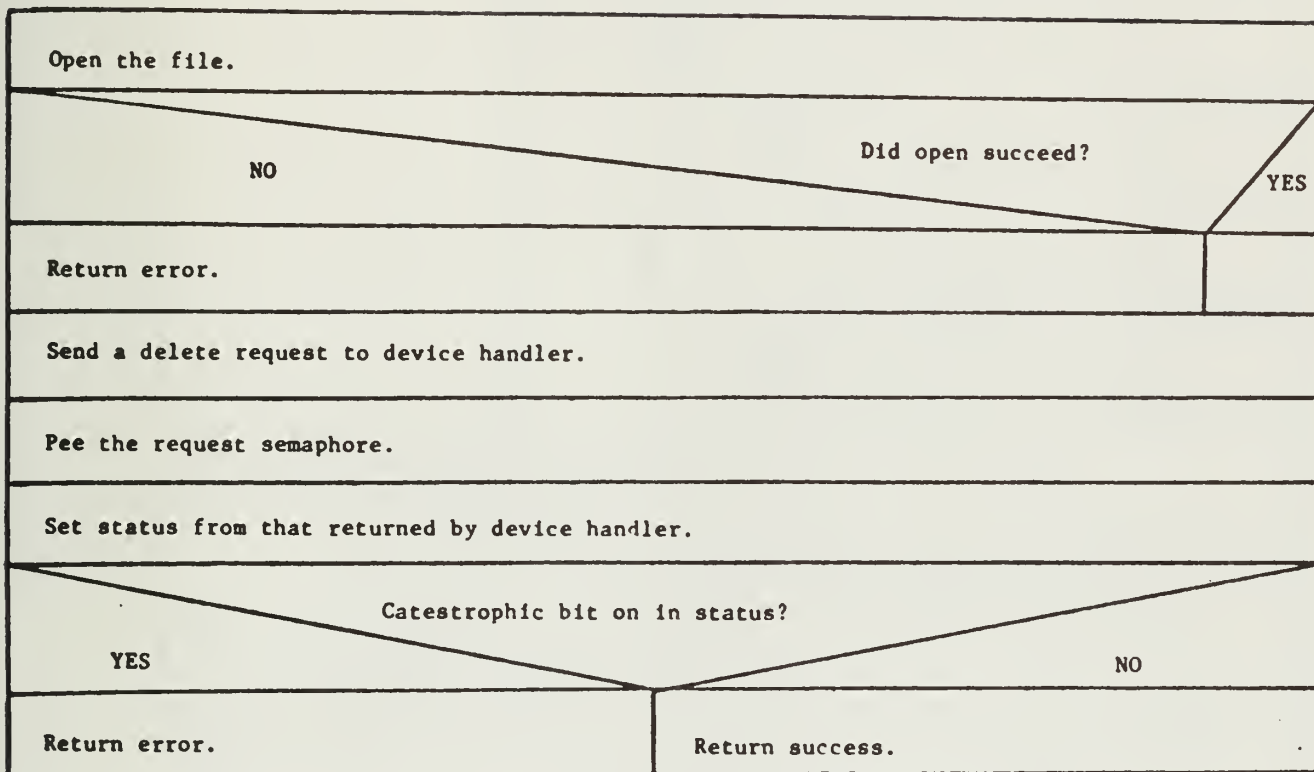
Allocate room for the stack.	
Did the alloc work?	
NO	YES
	Get pointer to the bottom of the stack.
	Store parm in the last word of the stack.
	Set up a register save area just above the bottom of the stack; put dummy values in the stack and environment regs; make it look like the process was called by suicide, and zero all the other regs.
	Set up the stack base: set the guardwork, priority and stack size; point the B7 entry at the register save area at the bottom of the stack, point the B5 entry at the main procedure for the process.
	Put the new process on the READY_Q.
Return the value returned by alloc.	

NOTE: This version of creep is specific to the Level 6.

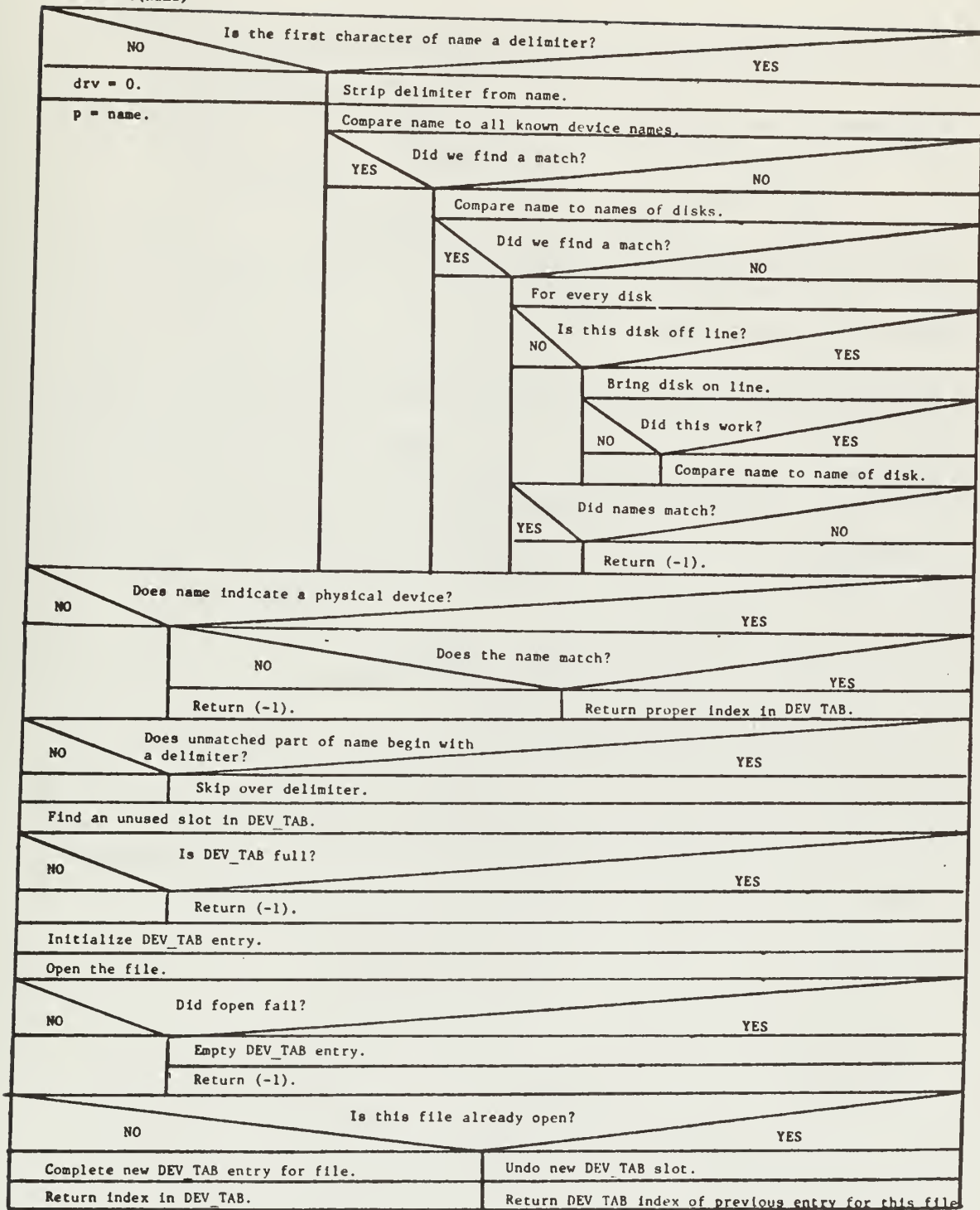
cvb(ptr, length, value_ptr)



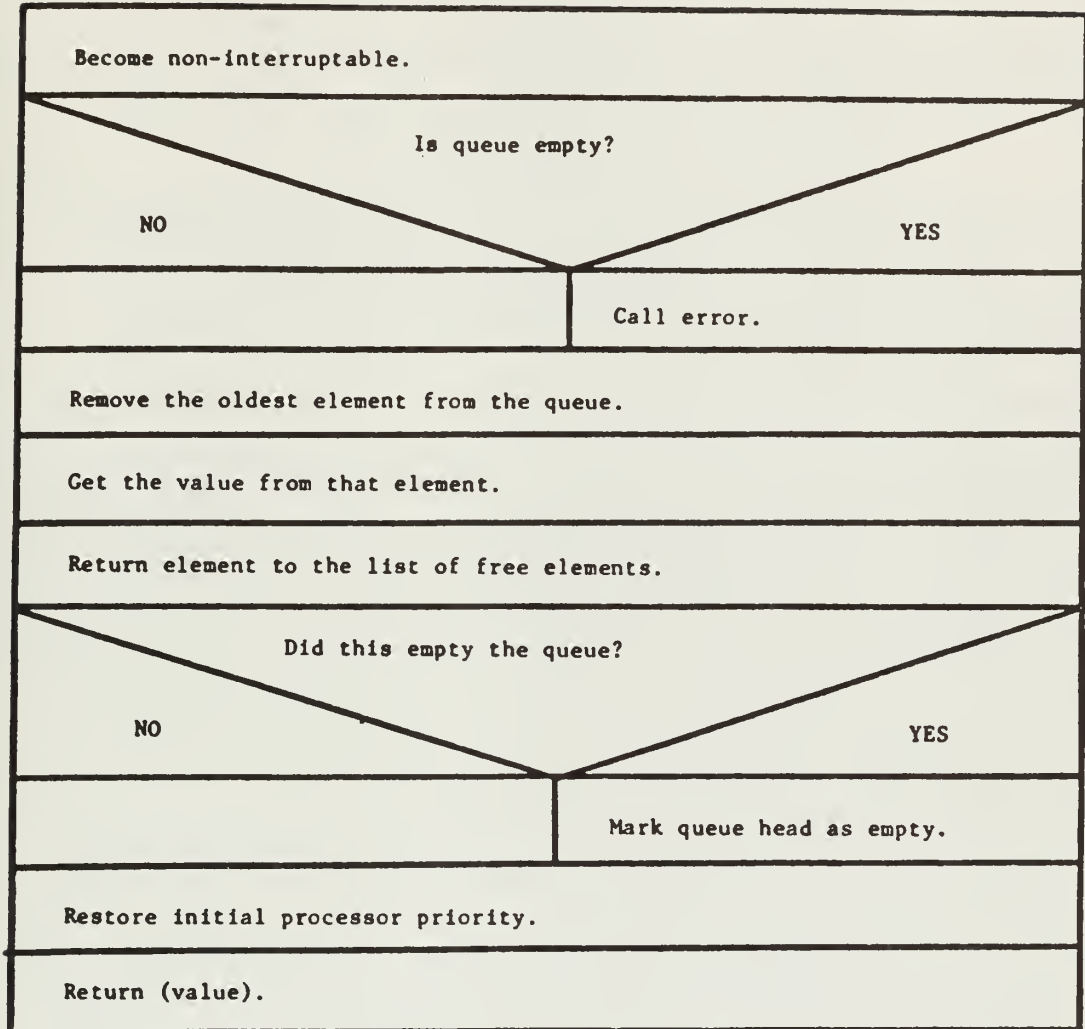
delete (filename, status)



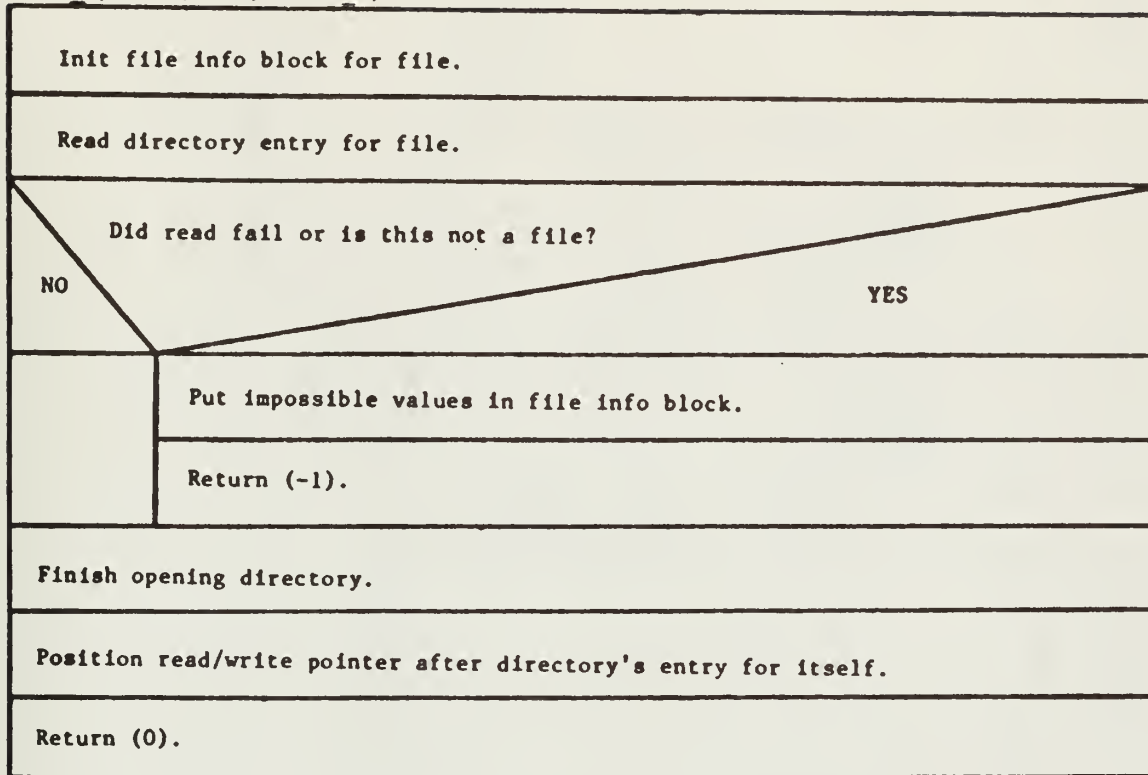
determine(name)



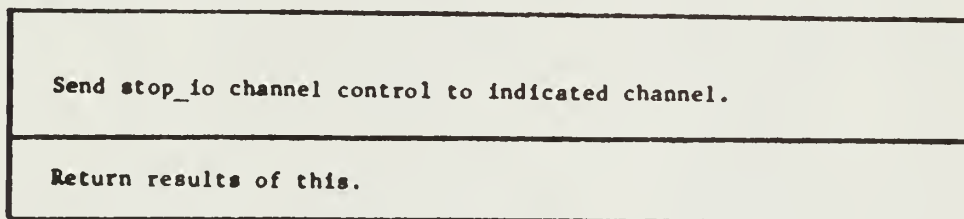
deq(q_ptr)



`dir_open(ev, fib, index b)`



`disablo_io(chan)`



Do forever

Input next request from DSKQ.

Decode command from request block.

Set user's status word to 0.

Switch (command)

open

close

read

write

peek

flush

delete

default

Close file.

Did close fail?

Set error bits in user's status.

Read data from file into user's buffer.

Did read fail?

Set error bits in caller's status.

Write data from user's buffer to file.

Did write fail?

Set error bits in caller's status.

Flush in_memory buffer.

Did flush fail?

Set error bits in caller's status.

Delete file.

Did delete fail?

Set error bits in caller's status.

Set reject reg bit in caller's status

Vee requesting process.

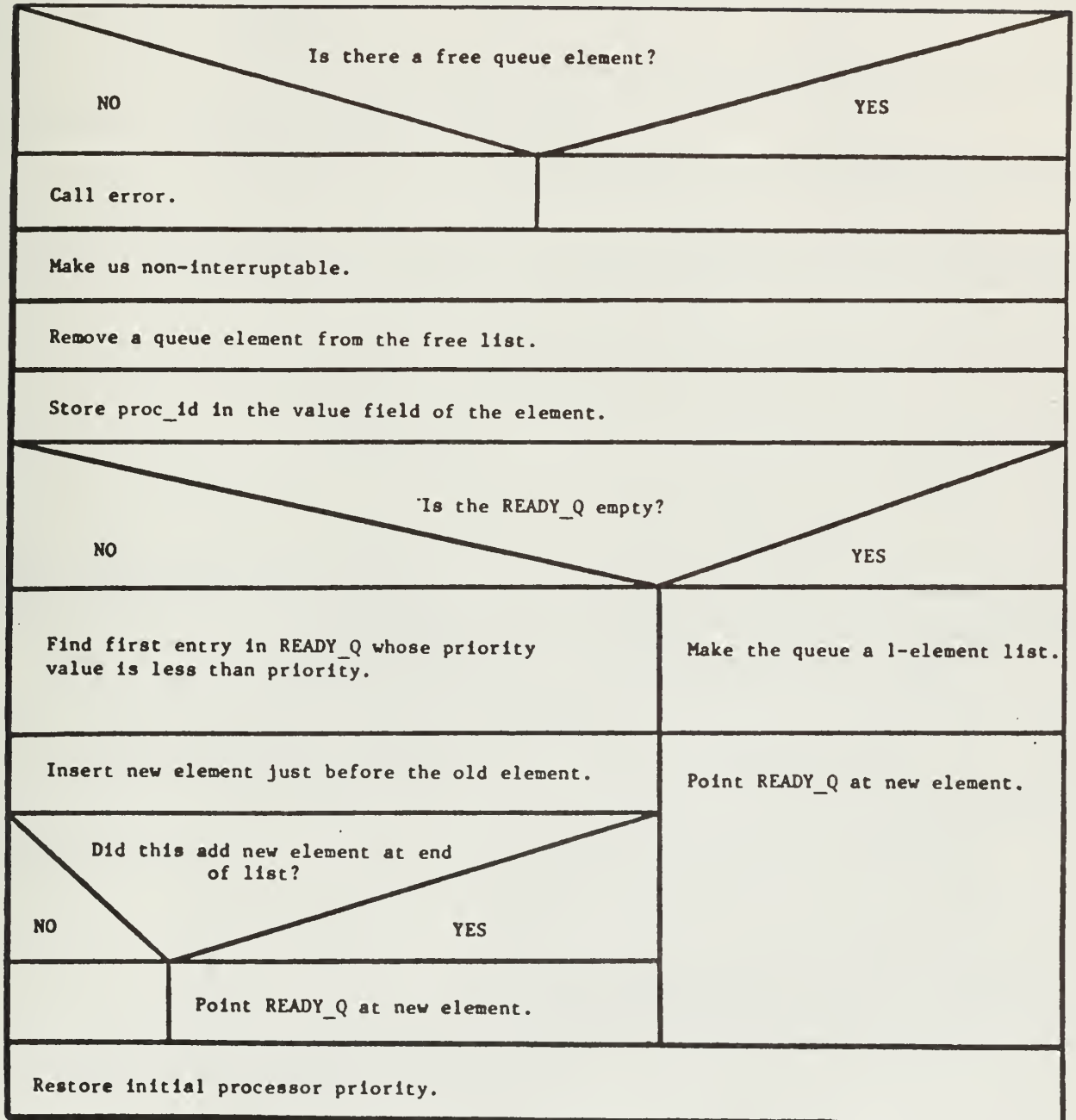
enable_io(chan, dev)

Is this an output channel?	
NO	YES
In_out = 0.	In_out = 1.
Addr = starting address in MLCP of the CCP for this channel.	
Flag = results of sending first byte of addr to the appropriate LCT byte.	
NO	Flag indicates error? YES
Flag = results of sending second byte of addr to the appropriate LCT byte.	
NO	Flag indicates error? YES
Flag = results of sending start_io channel control word to the channel.	
Return (flag).	

enq(q_ptr, value)

Is there a free queue element?	
NO	YES
Call error.	
Become non-interruptable.	
Remove a queue element from the free list.	
Store value in the element.	
Is the queue empty?	
NO	YES
Add the queue element to the end of the queue element list.	Make the queue a one-element list.
Point the queue head at the new element.	
Restore initial processor priority.	

enq_RQ(proc_id, priority)



entry

Compute the high end of usable memory.
Set up the stack registers.
Put address of high end of memory onto stack.
Jump to start up.

erase (x, y, vector, count, flag)

Compute any shifting required to do addressing on 16-dot boundaries.			
Is it a single vector written many times?			
YES			NO
Is shifting required?			
YES	NO		
Write top part of vector.	Write out the vector.		
Write bottom part of vector.			
		Rsrv_pnl.	
		Is shifting required?	
		YES	NO
		Copy vectors into local buffer and write out 18 at a time, shifting to write lower parts of vectors.	
		Copy vectors into local buffer and write out 18 at a time, shifting to write upper parts of vectors (or all of vectors).	
		Rls_pnl.	

NOTE: This version of erase is specific to the Level 6 IT.

erase (X, Y, vector, count, flag)

Compute the amount of shifting required to do the erases on 15-dot boundaries	
Set up panel registers for erase	
Do count times	
Yes	Is it an erase of a single vector? No
data = vector	data = next word pointed at by vector
Yes	Is shifting required? No
Erase the lower part of vector "data" on screen	
Erase upper part (or all of) vector "data" on the screen	
Increment the X panel address	

NOTE: This version of erase is specific to the 1531-11 IT.

`error(err_no)`

Become non-interruptable.
Print an error message, depending on the value of <code>err_no</code> .
<code>halt()</code> .

`fclose(ev, fib)`

Flush the <code>in_memory</code> index block for the file.	
Has the file gotten bigger?	
NO	YES
	Update directory to indicate new size.
Flush the <code>in_memory</code> data buffer.	
Mark <code>ev</code> and <code>fib</code> as unused.	
Return (0).	

fdelete(ev, fib)

Flush in_memory copy of file's index block.

Did flush fail, or is this the root directory?

NO

YES

Return(-1).

Mark all blocks used by the file as free.

Did truncate fail?

NO

YES

Restore in_memory copy of freemap.

Return(-1).

Mark index block for file as free.

Flag = false.

Flag = results of opening directory containing file.

Is flag set?

YES

NO

Flag = results of seeking to directory entry for file.

Is flag set?

YES

NO

Read directory entry for file.

Did read fail?

NO

YES

Flag = true.

Is flag set, or is this a protected file or a directory?

NO

YES

Restore in_memory copy of freemap.

Return(-1).

Mark directory entry on disk for file as empty.

Did write fail?

NO

YES

Restore in_memory copy of freemap.

Return(-1).

Clean up in_memory data buffer.

Did this work?

NO

YES

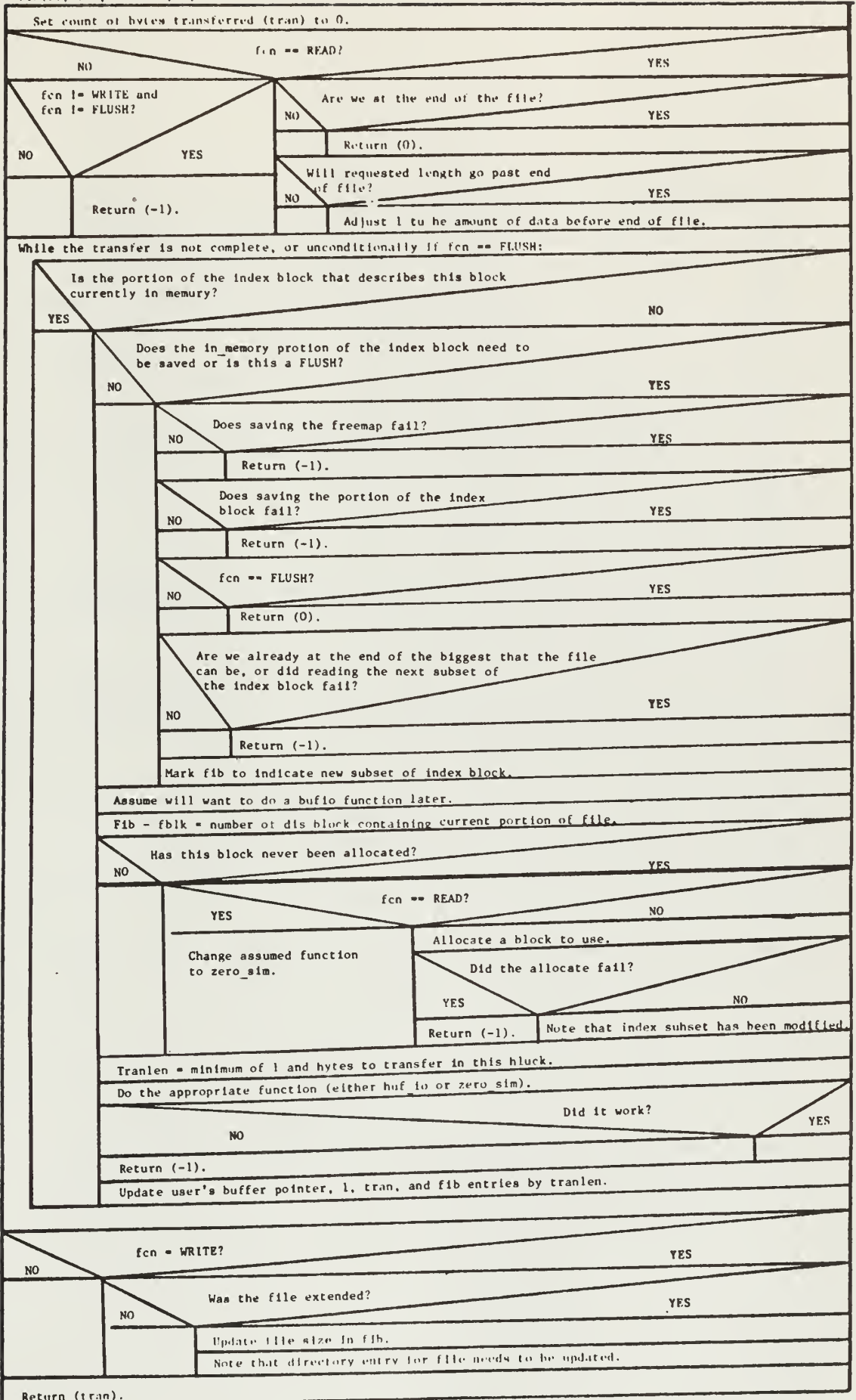
Return(-1).

Return(0).

`first_block`

Jump into the middle of block - to the point where it picks up a process from the ready queue.

file(ev, fib, userbuf, l, fcn)



fixup(reserve_size)

Save initial values of R5, R6, and memory location 4.

Put the address of trap_catcher in memory location 4.

For (R0 = 0;; R0 +=1024)

Try accessing memory location whose address is in R0 (will goto trap_catcher when try to access a non-existent location).

Count_down:
Decrement R0 and try accessing that location.

Decrement R0 by 1.

Subtract reserve_size from R0.

Copy R0 to R5.

R6 = R0 - the initial difference between R6 and R5.

Restore the initial value in memory location 4.

Make interruptable.

Clean up the stack.

Return to caller.

Trap_catcher:
Put the address of count_down on the stack.

Return from trap.

flush(device_id, st_ptr)

Is device_id out of range?	
YES	NO
Return error.	
Does this process own the device?	
NO	YES
Return error.	
Send a "flush" message to the device handler process.	
Pee the request semaphore.	
Set status for return.	
Is catastrophic bit on in status?	
YES	NO
Return error.	Return no error.

fdopen(name, lib, drv, root)

Dev = address of buffer structure for this drive.

Loop forever

Open file whose index block is root as a directory.

Did dir_open fail?

NO

YES

Return(-1).

Have we reached the end of the name?

NO

YES

Set file offset in fib to 0.

Return(0).

For (off = size of a dir_entry structure;
off → size of dir_entry structure)

Read next directory from directory.

Did read fail?

NO

YES

Return(-1).

Is this directory entry empty?

NO

YES

Continue.

New = results of using pathname to compare name to name for this entry.

Did they match at all?

NO

YES

Did they match up to a delimiter in name?

NO

YES

Is this entry a directory?

YES

NO

Return(-1).

Update name to point to character after delimiter.

Update root to be index block for this entry.

Break.

Did they match up to the end of name?

NO

YES

Is this entry a directory?

NO

YES

Update root to be index block for this entry.

Update name to point to end of name.

Break.

Return results of doing xopen on this entry.

free(size, addr)

Find the first entry in CORETAB whose address is less than addr. This may be the entry after the last used entry.			
NO		Is the preceeding entry contiguous with the one being freed?	
YES			
NO		Is the entry being freed con- tiguous with the succeed- ing one?	
YES		Add size to size of previous entry.	
NO		Does this make previous entry contiguous with this one?	
YES			
Copy the succeed- entries down a slot.	Move address of this entry to the begin- ning of the freed piece.	NO	YES
		Add size of this entry to previous one.	
Insert the piece to be freed into the table.	Add size to the size of this entry.	Move rest of table up one slot to delete this entry.	

fseek(fib, off, type)

Is offset in blocks or bytes?		
Blocks	Bytes	
b = number of blocks.	Is position absolute?	
	NO	YES
o = 0.	Treating off as a signed integer, convert it to a number of blocks b and an offset o into the last block.	
	Treating off as an unsigned integer, convert it to a number of blocks b and offset o in the last block.	
	o less than 0?	
	NO	YES
	b = -1 Add block size to o to make it positive.	
Switch(type)		
0,3	1,4	2,5
Set the read/write pointer for this file to block of b, offset of o.	Move read/write pointer to end of file.	
	Add block offset of b and byte offset of o to read/write pointer.	
	Is byte offset more than one block big?	
	NO	YES
		Subtract block size from byte offset.
	Increment block count by one.	

ftrunc(ev, fib)

Write out the index block for this file.

Max = number of blocks in file.

Has the last block been allocated?

NO

YES

Decrement max.

Indx = number of disk block that is disk block for this file.

Are there any data blocks in this file?

NO

YES

Return(0).

For every block in file:

Read in the number of disk block for this block.

Did read work?

NO

YES

Return(-1).

Free the disk block with this number.

Zero the index block.

Set size of file to 0.

Set necessary flags in fib.

Return(0).

`get_charset()`

Return (CS_ID)

`get_cursor(x_ptr, y_ptr)`

Convert current cursor position from dots to characters,
using the size of characters in the current char set.

Assign dimensions into *x_ptr and *y_ptr.

NOTE: This version of `get_cursor` is specific to the LSI-11 1T.

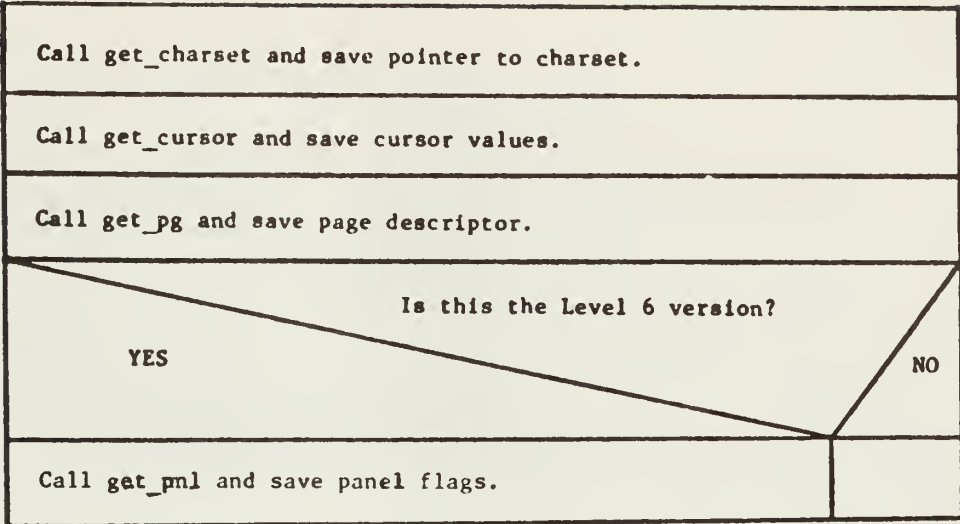
`get_cursor (x_ptr, y_ptr)`

Read cursor from the display head.

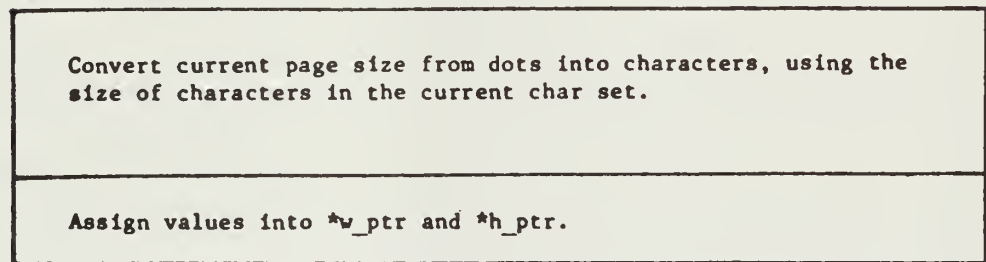
Convert values returned to character offsets.

NOTE: This version of `get_cursor` is specific to the Level 6 1T.

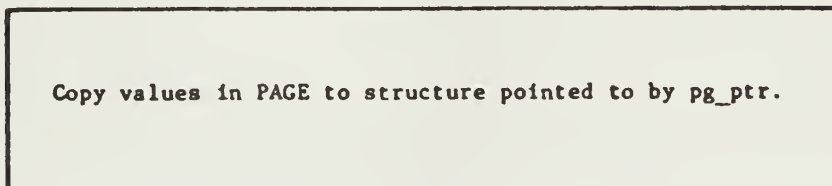
get_env (env_ptr)



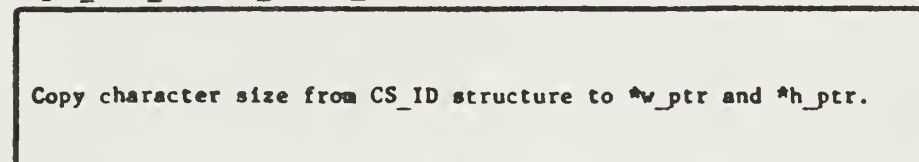
get_page_size(w_ptr, h_ptr)



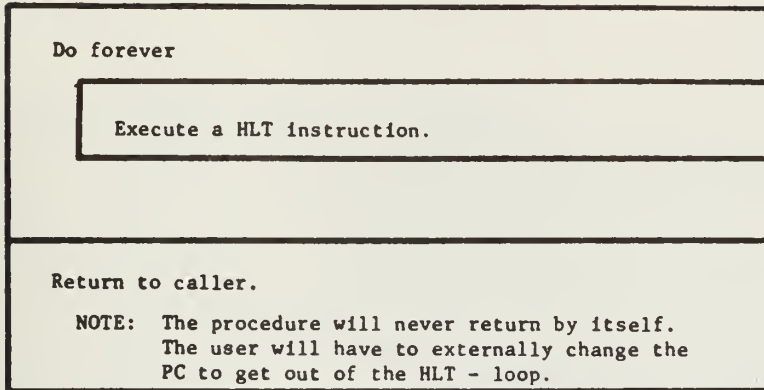
get_pg(pg_ptr)



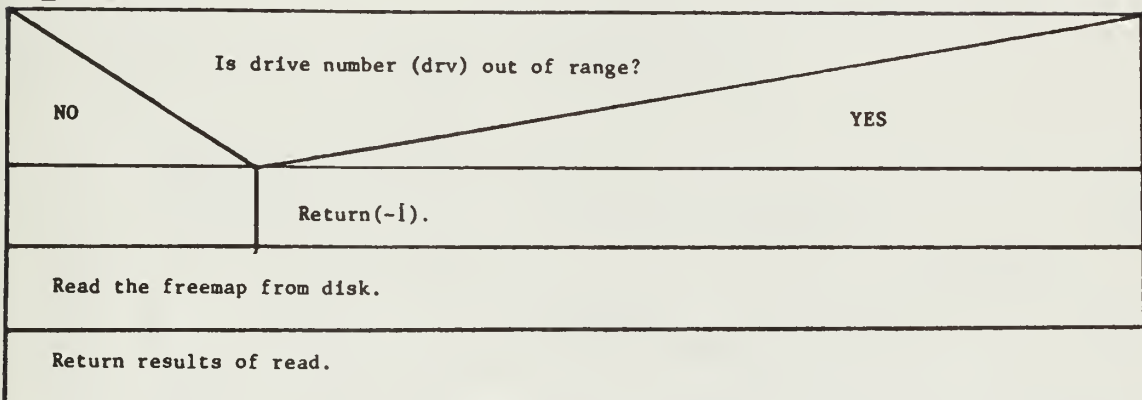
get_size_chars(w_ptr, h_ptr)



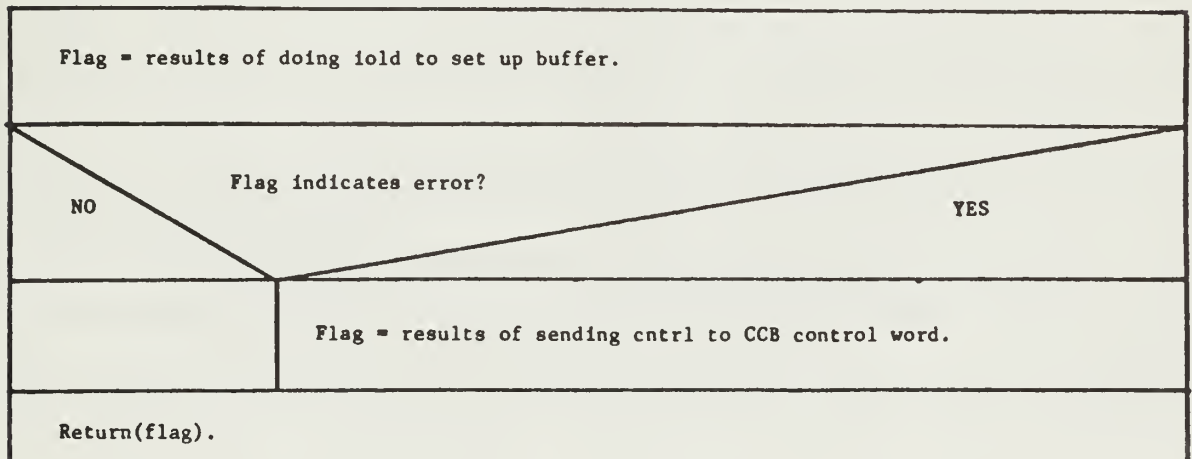
halt



1_freemap(drv)



init_ccb(chan, buf, buf_size, cntrl)



get_token(buf_ptr, tok_ptr, delim_ptr)

Get index of first char in buf that is not a delimiter, and store it in count.

Are all the characters delimiters?

NO

YES

Null terminate buffer.

Return (0).

For every char in buffer, starting with first delimiter

For every char in delimiter string

Char in buffer == char in delimiter?

NO

YES

Terminate outermost for_loop.

Copy character from buffer to token.

Increment count.

Null terminate token buffer.

Return count of characters in token and count of delimiter characters scanned at beginning.

init_drive(drv)

NO	Is the drive number (drv) out of range?
	Return (-1).
Set up dib entry for drive to indicate that drive is open.	
Open root directory of drive.	
NO	Did open fail?
	Return (-1).
Read entry in root directory describing itself.	
Read freemap for disk.	
NO	Did either of the reads fail?
	Indicate disk is off line.
	Return (-1).
Return (0).	


```
init_fib(fib, index_b, off)
```

Assign appropriate values to elements of file information block.

Return (0).

```
io_init()
```

Put the address of each device's handler's input queue into DEV_TAB.

Do for each device

Put the address of this device's request semaphore into the request block for this device.

```
init_pnl()
```

Call Z80_LD to get a pointer to the Z80 microcode

Set up so that writes go to all remote display heads

Is there an alternate charset loaded?

YES

NO

Flag the charset as not loaded

Has the microcode already been loaded?

NO

YES

Write the microcode to the Z80 panel controller

Free the space occupied by the microcode in the Level 6 memory.

Initialize the Z80's internal variables.

index(in_str, of_str)

lth_of = length of "of" string.

Point p1 at in_string and p2 at of_str.

For every character in "in" string

Does char in "in" string == char in "of" string?

NO

YES

Is next char of "of" string ending null?

NO

YES

Point p1 and p3 into "in" string, where match would start to end here.

Compare the characters pointed at by p2 and p1, ending when they differ, or when hit end of "of" string.

Did all chars match?

NO

YES

p1 = p3.

p2 = beginning of of_str.

Return index into in_string of beginning of match.

Increment p1.

Return (-1).

b_driver()

Enable keyboard input.

Do forever

Read a word from the input queue.

Is the word odd?

YES

NO

Request_code = "data."

The word is a pointer to a request block containing the request code.

Is high byte of the word a CR?

YES

NO

Input data character is newline.

Input data character is in the high byte of the word.

Switch on request code.

peek	close	open	flush	setmode	data	read	default
Put count of characters in buffer into request.block.	Set "open" flag off.	Set "open" flag on.	(See kb_driver: flush_type)	(See kb_driver: setmode_type)	(See kb_driver: data_type)	(See kb_driver: read_type)	Error: Reject the request.
Vee the semaphore.	Vee the semaphore.	Set x - y address for echoing to default. Vee the semaphore.					

NOTE: This version of kb_driver is specific to the LSI 11 II.

kb_driver

Enable keyboard input.

Do forever

Read a word from the input queue.

Is it a known special code?

YES

NO

The word read is the request code.

The word read is a pointer to a request block containing the request code.

Switch on request code.

peek

close

open

flush

setmode

overflow
or data

read

default

Put count of
characters in
buffer into
request block.Set "open"
flag off.Set "open"
flag on.Vee the sema-
phore.Vee the sema-
phore.Vee the sema-
phore.

(See kb_driver: flush_type)

(See kb_driver: setmode_type)

Is a read
outstanding?

NO

YES

Break.

Use the request
block from pre-
vious read.

Is data available?

YES

NO

Copy data in-
to user's buf-
fer.Save pointer
to the request
block.Vee user's
semaphore.Error -
Reject the
request.

NOTE: This version of kb_driver is specific to the Level 6.

kb_driver: data_type*

Is the keyboard open?			
YES		NO	
Is there a saved up read?		Ring_bell.	
YES	NO		
Put input character into user's buffer.	Is there space in buffer?		
Vec user's semaphore.			
	YES		NO
Reset "saved read" flag.	Put the character into the buffer.		
Are we echoing?			
YES		NO	
Print the character using put_sacii.			

*This is not a procedure. It is one case in kb_driver.

kb_driver: flush_type

Empty the input buffer.	
Is a read pending?	
YES	NO
Set data length to zero and reject the saved read request.	
Vee the user's semaphore.	

NOTE: This is not a procedure. It is one case in kb_driver.

kb_driver: read_type*

Is the buffer empty?	
NO	YES
Compute length to give the user.	Save the read request.
Copy characters into user's buffer.	
Vee user's semaphore.	

*This is not a procedure. It is one case in kb_driver.

kb_driver: setmode_type

Switch on type of setmode.			
echo_on	set_xy	echo_off	default
Set "echoing" flag on.		Set "echoing" flag off.	Set "reject request" in request block.
Set x - y coordinates.			
Vas user's semaphore.			

kill(proc_id)

Does id indicate a valid process?	
NO	YES
Return(-1).	
Set the guardword to indicate that the process should be killed the next time it is scheduled.	
Return(0).	

ld_cs()

CS ID != CS_LAST (1.e. has the character set been changed)?	
YES	NO
Is this charset resident in the display head?	
NO	YES
Is there an alternate charset already loaded?	
YES	NO
Flag the old alternate cs that it is no longer loaded.	
Rsrv_pnl.	
Allocate space for the effector table as "variable" number 3.	
Write the effector table.	
Allocate space for the characters as "variable" number 2.	
Write the characters (the bytes of each word must be swaped before writing them).	
Flag the current character set as the currently loaded alternate.	
Rls_pnl.	
Write out a new charset descriptor (specifying the size of the characters and which "variables" to use for the effector and character tables).	

ld_page()

Has the page been changed since last time we were here?	
YES	NO
Format a page descriptor for the Z80.	
Write it out.	

ldiv(hi, lo, d)

Copy the pair (hi, lo) to registers R0 and R1.

Do a double word divide on (R0, R1) by d.

Return the quotient.

NOTE: This version of ldiv is specific to the LSI 11 IT.

ldiv(hi, lo, d)

Copy the pair (hi, lo) to registers R6 and R7.

Do a double word divide on (R6, R7) by d.

Return the quotient.

NOTE: This version of ldiv is specific to the L6 IT.

`ln_xpand(out, in, p1, p2, p3, . . .)`

`Length = 0.`

`Next_parm = address of first parameter (p1).`

While there is still something in the input format specification

Is the next char in the input format specifier a parameter replacement indicator?	
NO	YES
Copy the char from the input buffer to the output buffer.	<code>ptr = parm_xpand</code> (pointer to format specifier, and <code>next_parm</code> , and <code>skipped</code>).
Move the input buffer over one character.	Increment in pointer by <code>skipped</code> .
Increment length by one.	Copy the string pointed to by <code>ptr</code> into the output buffer, and increment length by the length of this string.

Null terminate the output string.

Increment length to include the trailing null.

Return length.

`lrem(hi, lo, d)`

Copy the pair (hi, lo) to registers R0 and R1.

Do a double word divide on (R0, R1) by d.

Return the remainder.

NOTE: This version of `lrem` is specific to the LSI 11 IT.

`lrem(hi, lo, d)`

Copy the pair (hi, lo) to registers R6 and R7.

Do a double word division on (R6, R7) by d.

Return the remainder.

NOTE: This version of `lrem` is specific to the L6 IT.

`mfps()`

Return the value of the PSW.

NOTE: This version of `mfps` is specific to the LSI 11 IT.

`mfps()`

Pick up the system status word.

Decode the current level number from it.

Is this level greater than the non-interrupt-
able level for the L6?

NO

YES

lev = the non-interruptable
status word for LSI 11.

lev = 0.

Return (lev).

NOTE: This version of `mfps` is specific to the L6 IT.

mk_page(p_ptr, left, bottom, width, height)

Copy parameters to corresponding positions in structure pointed to by p_ptr.

mtps(prio)

Does prio have the LSI 11 non-interruptable bit set?		Do a LEV that will leave the activity bit set for the current level, and switch us to be working at the non-interruptable level.
NO	YES	
Pick up the hardware status word.		
Decode the current level.		
Are we currently interruptable?		
NO	YES	
Do a LEV that resets the activity bit for the non-interruptable level and schedules the next ready level.	Leave us at our current level.	

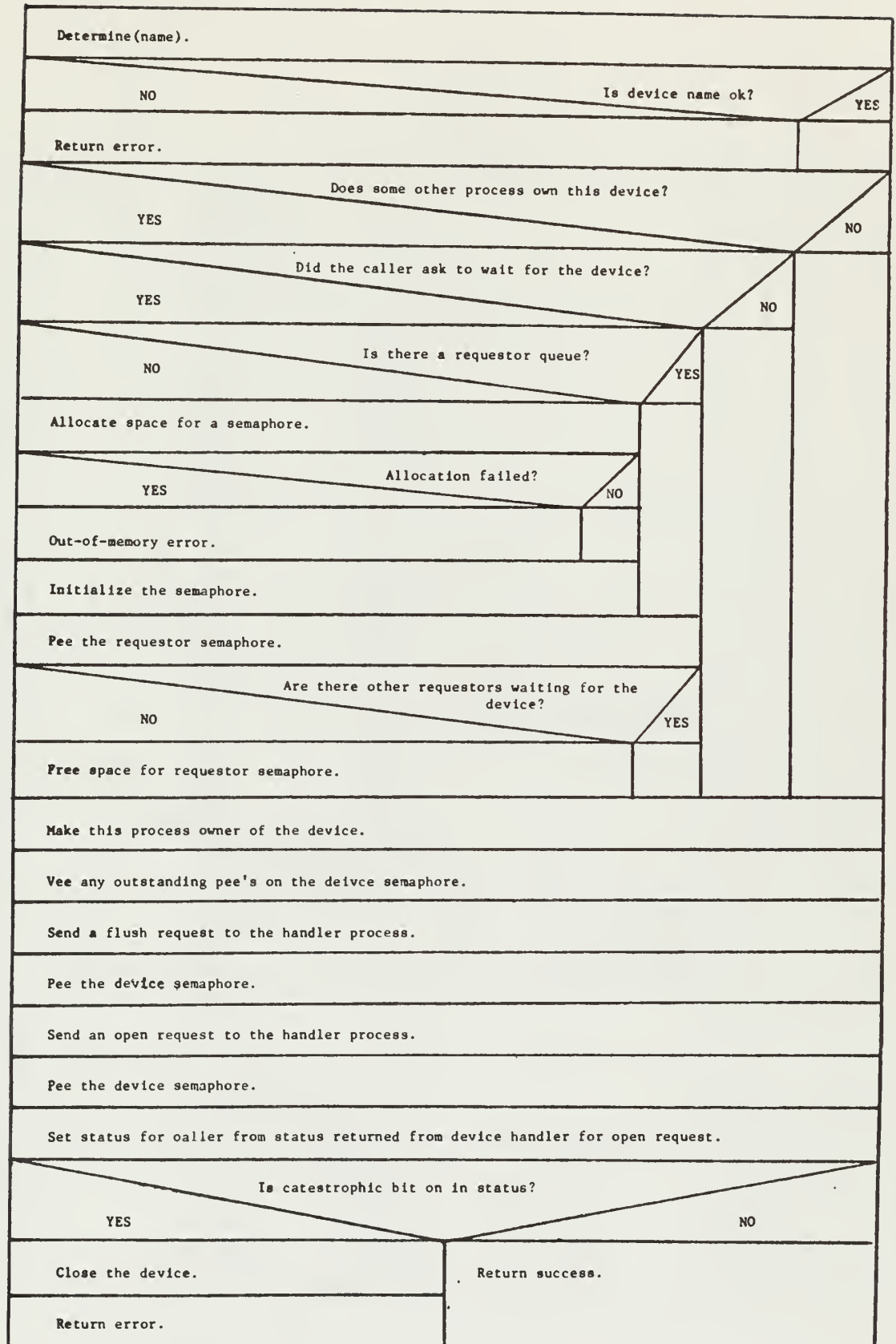
NOTE: This version of mtps is specific to the L6 IT.

mtps(prio)

Copy prio into the PSW.

NOTE: This version of mtps is specific to the LS11 IT.

open(name, flag & status)



parm_xpand(how, what, count)

Set word pointed to by count to be the number of characters in the format description for this expansion.					
Switch on type of expansion.					
type 'g'	type 'c'	type 'd'	type 'l', type 'u'	type 'o'	type 'x'
Pick up char ptr pointed to by what.	Copy char pointed to by what to workspace buffer.	Is parm less than 0? YES NO		Put a 'o' in the buffer.	Put an 'x' in the buffer.
Increment what to point to next parameter in caller's list.		Put a 'l' in buffer.	Do a str num on parm, with a base of 10, into buffer.	Do a str num on the parm, with a base of 8, into the buffer.	Do a str num on the parm, with a base of 16, into the buffer.
		Copy parm into temp.			
Return char ptr.		Do a str num on temp, with a base of 10, into buffer.			
Null terminate the string have built in the buffer.					
Move what to point to next parameter in caller's list.					
Return a pointer to the beginning of the workspace buffer.					

Copy the type character into the buffer.

Do a str num on the parm, with a base of 16, into the buffer.

Do a str num on the parm, with a base of 8, into the buffer.

Do a str num on parm, with a base of 10, into buffer.

Return char ptr.

Null terminate the string have built in the buffer.

Move what to point to next parameter in caller's list.

Return a pointer to the beginning of the workspace buffer.

pause()

Enqueue this process on the READY_Q.

Call block().

Return.

pathname(your, dir)

Remember where your string begins.

Compare characters in your and dir strings up to the end of dir and as long as the two strings match.

Did the strings differ before the end of the dir string,
or did the comparison end pointing at something other
than a NUL or delimiter in your string?

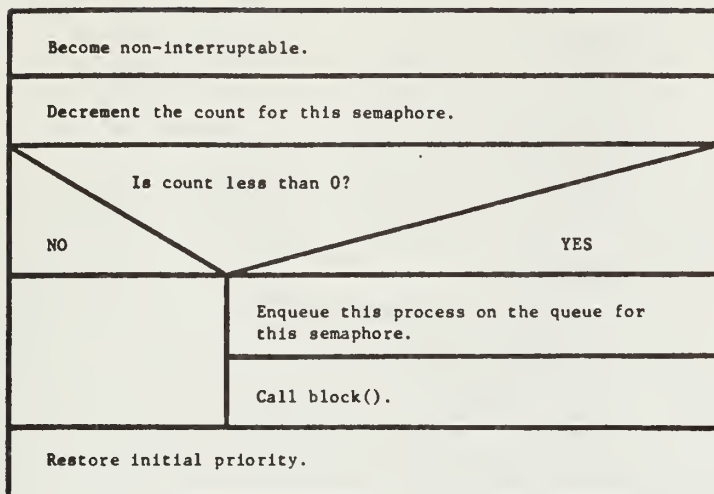
NO

YES

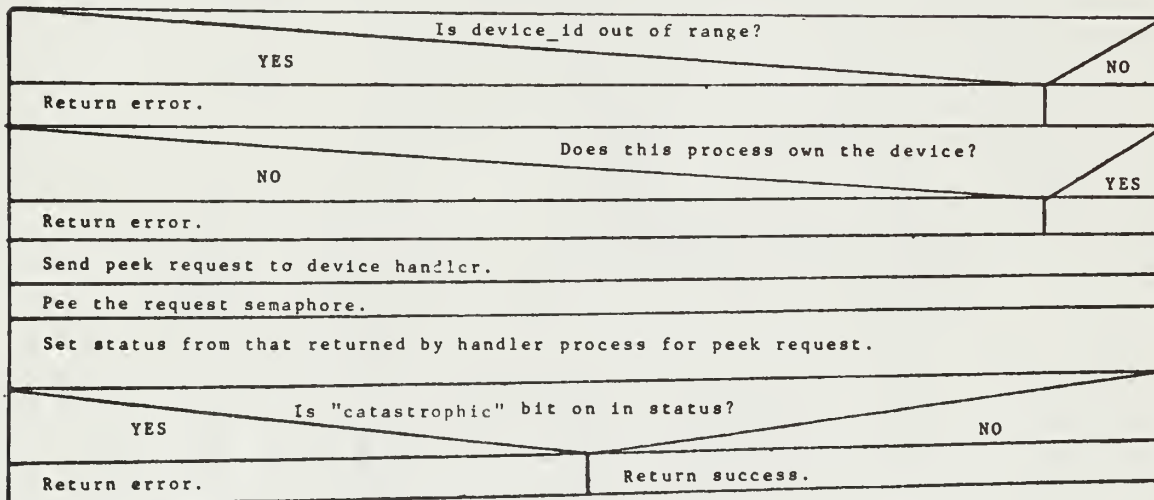
Return pointer to beginning of your string.

Return pointer to your string where comparison ended.

peek(sem)



peek(device_id, &status)



ph_driver()											
Ph_available = true.											
Saved_read = active_write = 0.											
While (true)											
Read an element from input queue.											
Is this a simple request?											
NO											
YES											
Blk_ptr = what read from queue.											
Command = what read from queue.											
Command = command in request block.											
Switch (command)											
open	close	read	write	peek	flush	data	overflow	write failed	done	status change	default
(See ph_driver: open_type)	(See ph_driver: close_type)	(See ph_driver: read_type)	(See ph_driver: write_type)	Calculate amount of data in our buffer, including trailing null if buffer is not empty.				Set catastrophic bit in writer's status.			Set reject_req bit in caller's status.
				Assign the value to caller's data_len.				Set writer's data_len to amount of data actually transmitted.			Vee requesting process.
				if no data and phone not available? YES Set catastrophic bit in caller's status. NO Vee requesting process.	(See ph_driver: flush_type)	(See ph_driver: data)		Vee writing process.		(See ph_driver: status_change)	
								Set active_write to 0.			

NOTE: This version of ph_driver is specific to the Level 4 IT.

Do "forever"

Is it a known special code?

No

```
request' code is decoded from value read from queue
'Value read is a pointer to a request block with the
request code
```

Do a switch on the request code

Do a switch on the request code											
open or close		read	write	peek	flush	data	done	up	down	default	
No	is phone available	See ph_driver:read_type	Is phone available	Set length equal to number of chars available, including trailing null, if buffer not empty	See ph_driver:flush_type	See ph_driver: data_type	Turn off output	Set phone available flag	Reset phone available flag	Reject the request	
			Yes	No							
Set error flag	See user's seraphore		Start a write	Assign the value to user's data_ len.	See ph_driver:flush_type	See ph_driver: data_type	See saved write's seraphore				
			Reject user's request	No data and phone not available							
				No							
				Yes							
				Set error flag for user							
				See user's seraphore							

ph_driver: data*

While there is a pending read, and there is more data in our buffer.

Copy the next character from our buffer into the reader's.

Put a null after char just copied.

Increment the reader's count of data in the buffer.

Is the buffer full, or was the last char copied a newline?

NO

YES

Vec reading process.

Set saved_read to 0.

Increment the next character index by 1, module our buffer size.

*This is not a procedure. It is one case in ph_driver.

ph_driver: close_type*

Is the phone available?

NO

YES

Flag = results of disabling phone input.

Is phone not available, or is flag set?

NO

YES

Set catastrophic and reject_req bits in caller's status word.

Vec requesting semaphore.

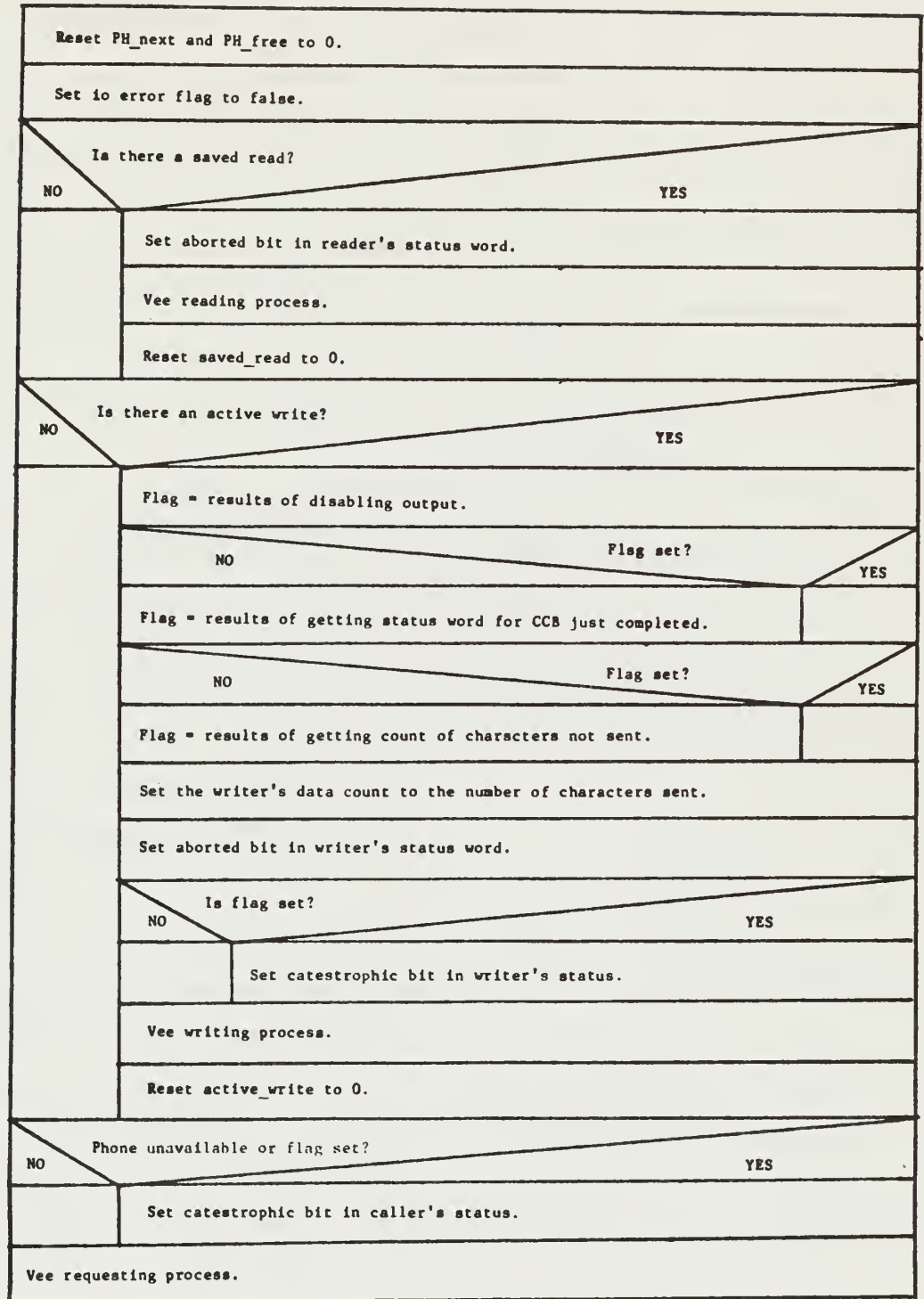
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: flush_type*

Stop any current output operation	
Empty the input buffer	
Is there a saved read?	
YES	NO
Reject the saved read	
Is there a saved write?	
YES	NO
Reject the saved write.	
Is the phone line down?	
YES	NO
Set error flag in request block.	
Vee user's semaphore.	

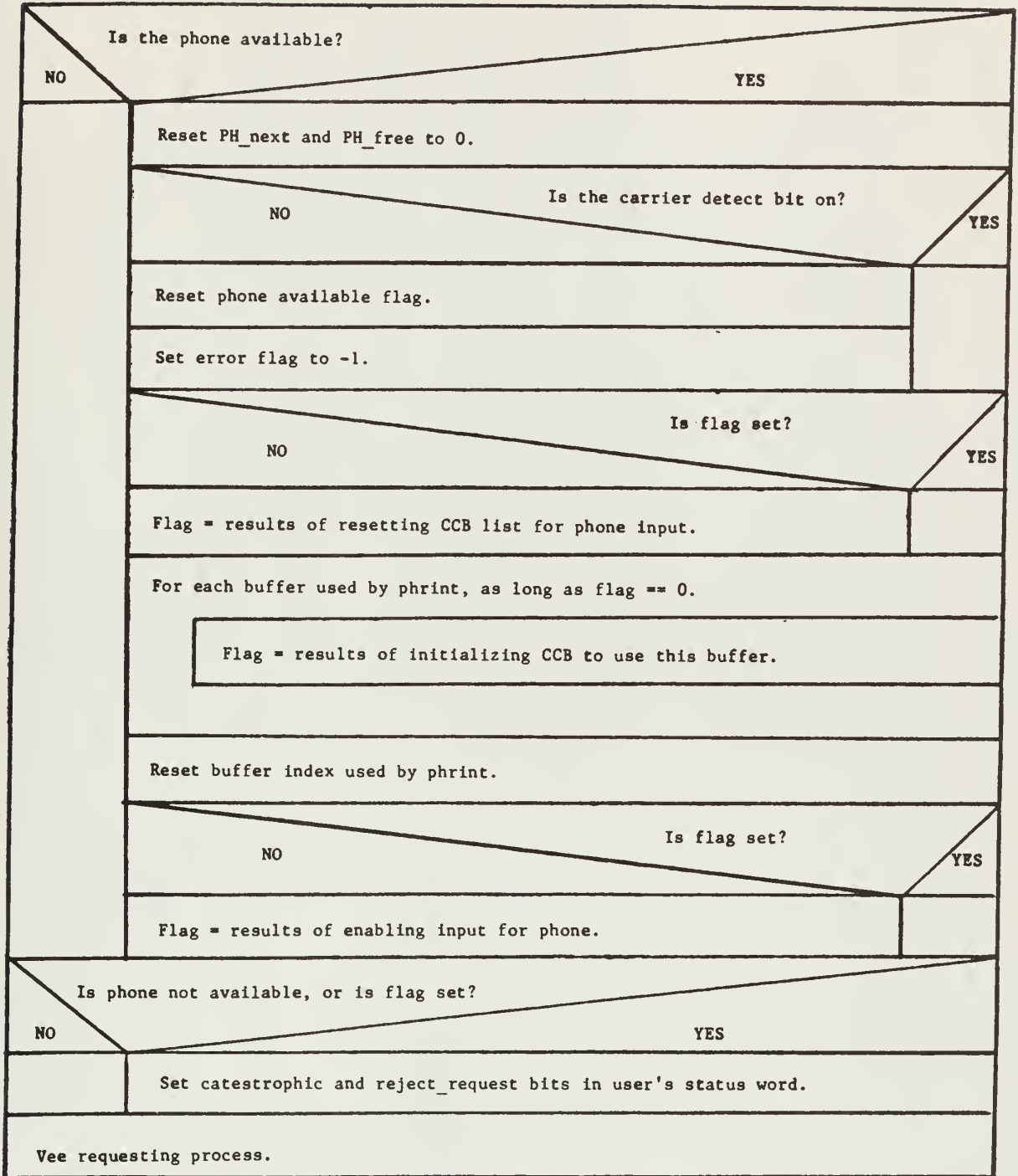
*NOTE: This is not a procedure. It is one case in ph_driver, and is specific to the LSI-11 IT.

ph_driver: flush_type*



*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: open_type*



*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: read_type*

Initialize blk_ptr > data_len to 1.	
Point out_ptr at user's buffer.	
Is there data in our buffer?	
NO	YES
Is the phone available?	Transfer data from our buffer to the user's until the user's buffer is full, or we are out of data or we hit a new line character.
NO	YES
Set catastrophic and reject_req bits in caller's status word.	Set saved_read = blk_ptr.
Set blk_ptr---data_len to the number of characters transferred, including the trailing null.	
Is user's buffer not full, and is the last char not a newline?	
NO	YES
Set saved_read = blk_ptr.	
Is saved_read zero?	
NO	YES
Vee requesting process.	

*This is not a procedure. It is one case of ph_driver, and is specific to the Level 0 II.

ph_driver: status_change*

Complement the phone available flag.		
NO	Phone available?	
	YES	
	NO	Pending read?
		YES
		Set catastrophic and aborted bits in reader's status.
		See reading process.
		Set saved_read to 0.
	NO	Active write?
		YES
		Turn off output.
		Input status for CCB just completed.
		Input number of chars not transmitted.
		Set catastrophic and aborted bits in writer's status.
		Set writer's data_len to count of chars actually sent.
		See writing process.
		Reset active_write to 0.

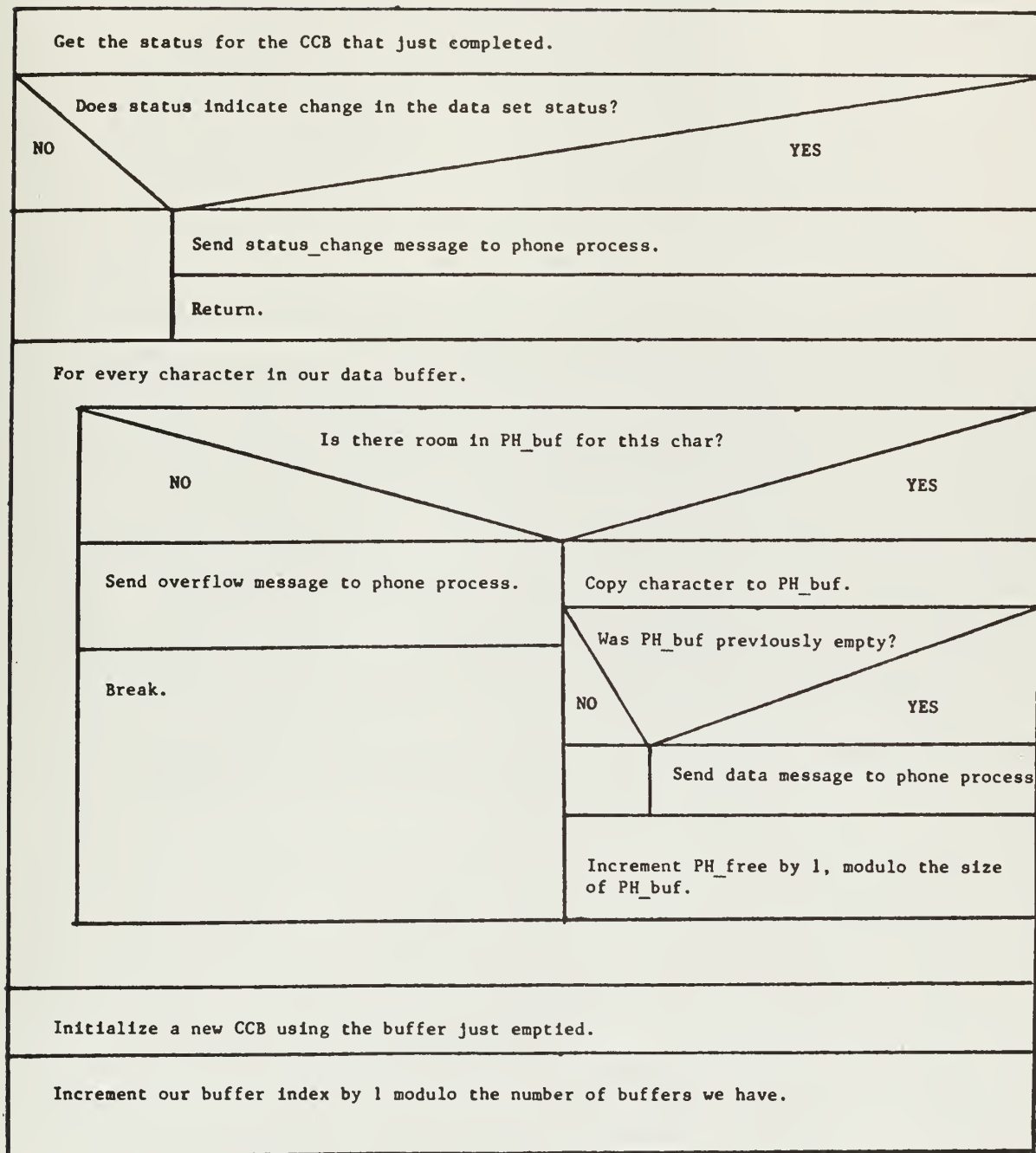
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

ph_driver: write_type*

NO	Is phone available?		YES
	Set active_write = blk_ptr.		
	Flag = results of sending reset CCB list command to output phone channel.		
	Is flag set?		YES
	Flag = results of initializing an output CCB to use caller's buffer.		
	Is flag set?		YES
	Flag = results of restarting phone output.		
	Is phone unavailable or is flag set?		
	Is flag set?		YES
	Reset active_write to 0.		
NO	Set catastrophic and reject_req bits in caller's status.		
	Vee calling process.		

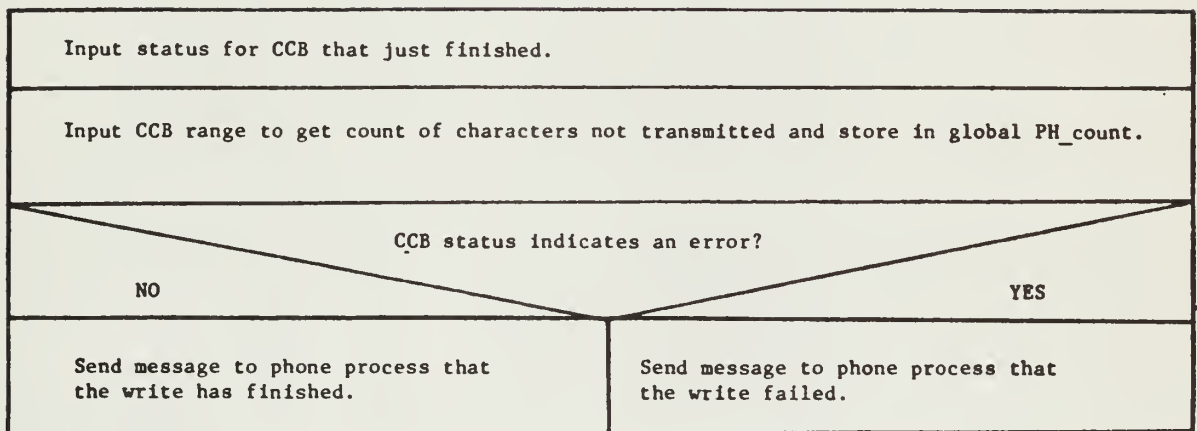
*This is not a procedure. It is one case in ph_driver and is specific to the Level 6 IT.

phrint()



NOTE: This version of phrint is specific to the Level 6 IT.

phxint()



NOTE: This version of phxint is specific to the Level 6 IT.

pp_read (code, buffer, length)

Rsrv_pnl().

Select any active panel to read from.

Do forever

Read controller interrupt level.

Make processor non-interruptable.

Is a keyboard or touch panel read in progress?

YES

NO

Set flag to make interrupt routine wake us up.

Attempt to initiate a read.

Did the read start?

YES

NO

Make processor interruptable.

280 interrupts enabled?

YES

NO

280 interrupts enabled?

Wait for read to complete.

Set flag for interrupt routine.

While read is still in progress

Null.

NO

YES

Pee the panel semaphore.

Make processor interruptable.

Pee the panel semaphore.

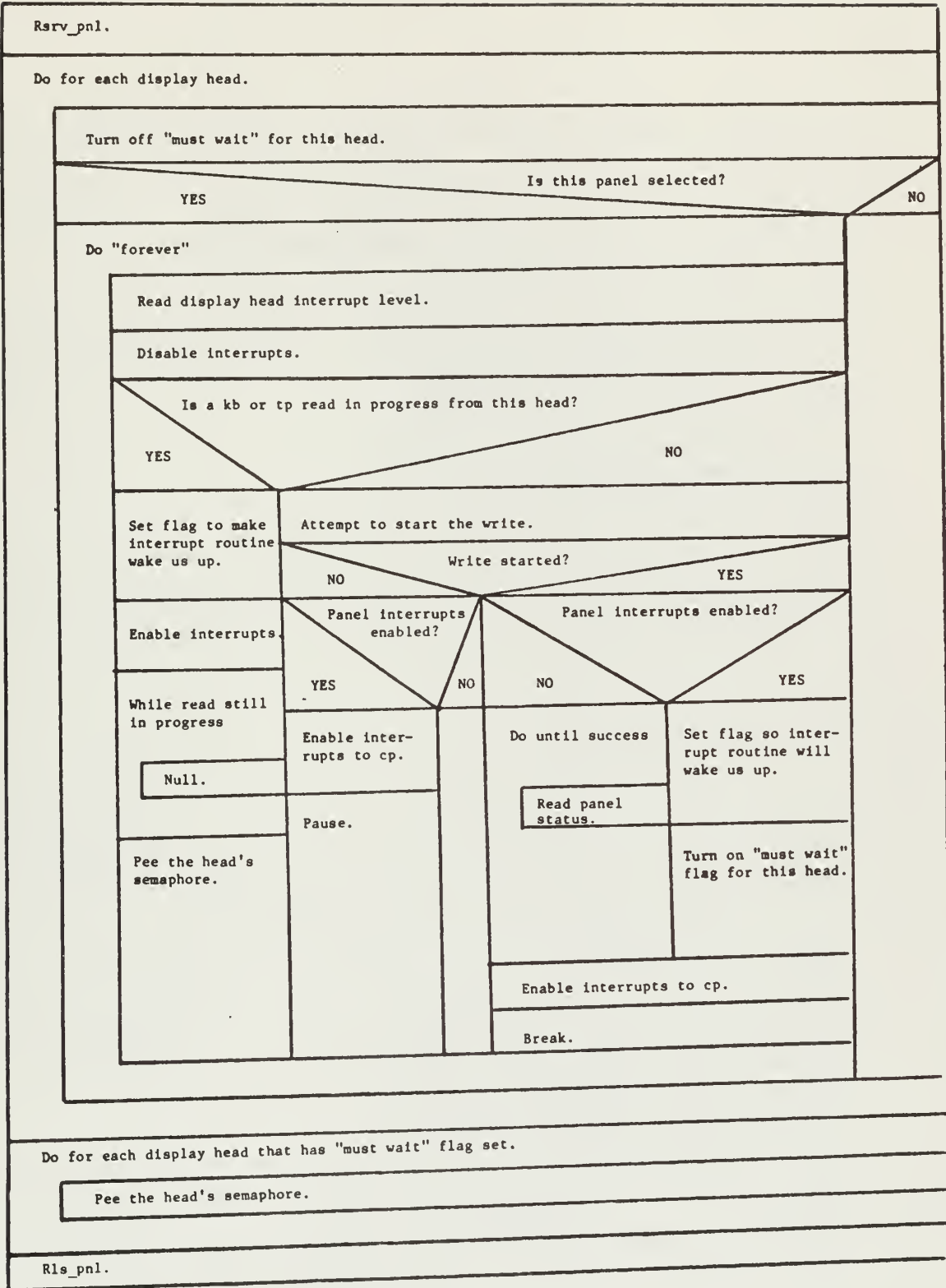
Make the processor interruptable.

Rls_pnl().

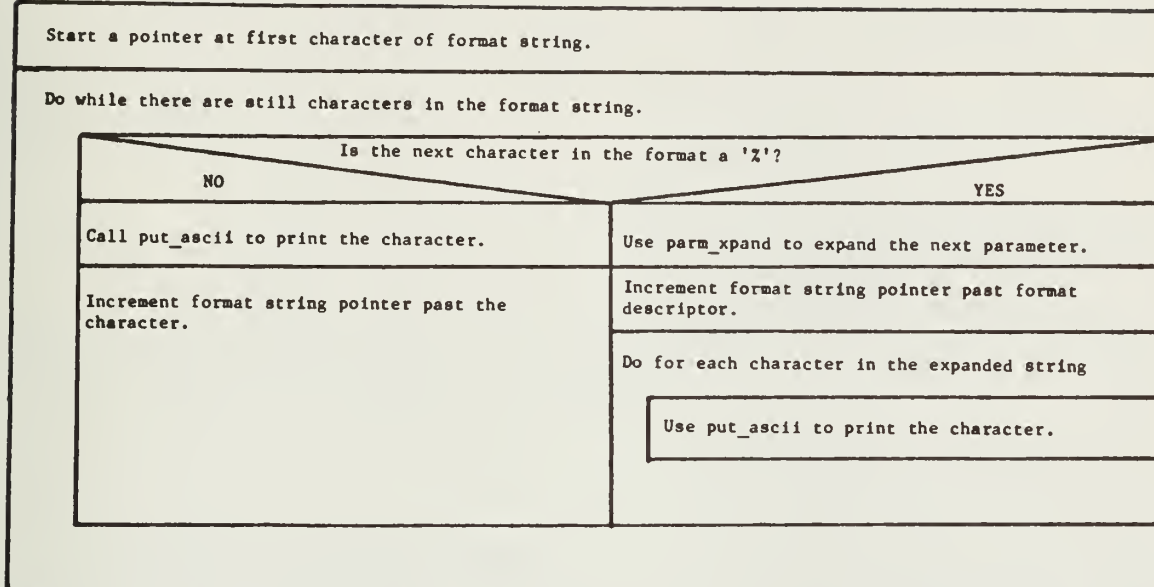
Return.

Pause.

pp_write() (code, buffer, count)

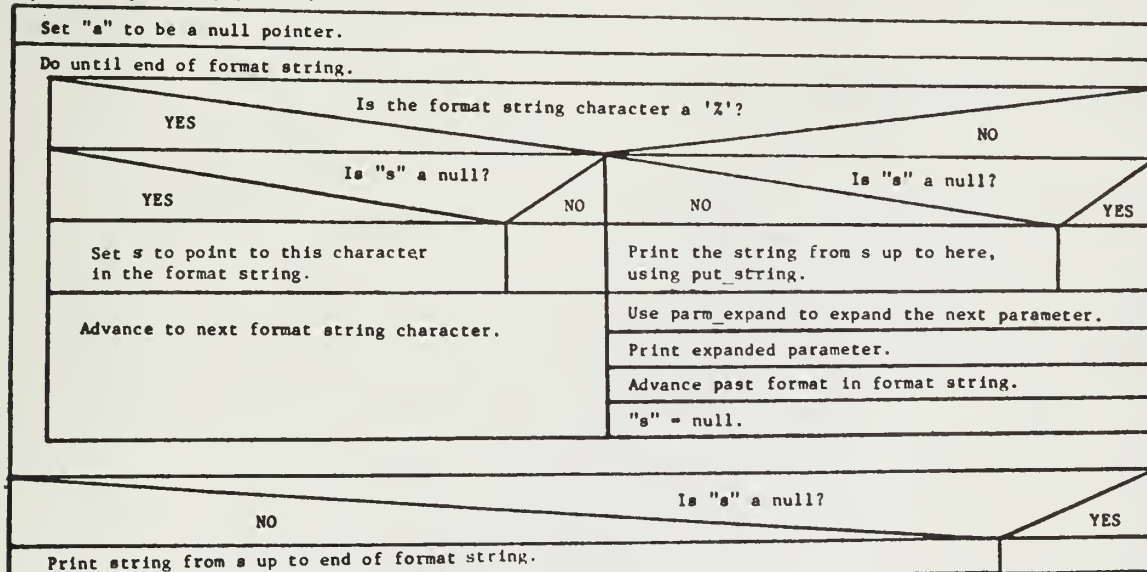



```
printf(format, p1, p2,.....)
```



NOTE: This version of printf is specific to the LSI 11 IT.

```
printf (fmt, p1, p2,.....)
```



NOTE: This version of printf is specific to the Level 6 IT.

put (x, y, vector, count, flag)

Compute any shifting required to do addressing on 16-dot boundaries.			
Is it a single vector written many times?			
YES		NO	
Is shifting required?		Rsrv_pnl.	
YES	NO	Is shifting required?	
Write top part of vector.		YES	NO
Write bottom part of vector.		Copy vectors into local buffer and write out 18 at a time, shifting to write lower parts of vectors.	
		Copy vectors into local buffer and write out 18 at a time, shifting to write upper parts of vectors (or all of vectors).	
		Rls_pnl.	
Write out the vector.			

NOTE: This version of put is specific to the Level 6 IT.

put (X, Y, vector, count, flag)

Compute amount of shifting required to do the puts on 16-dot boundaries	
Set up panel registers for write	
Do count times	
Is it a put of a single vector?	
Yes	No
data = vector	data = next word pointed at by vector
Is shifting required?	
Yes	No
Put lower part of vector "data" on the screen	
Put upper part (or all of) vector "data" on the screen	
Increment the X panel address	

NOTE: This version of put is specific to the LSI-11 IT.

put_ascii(char, 6x, 6y)

Is the x-cursor outside of the page?											NO
YES											
Put cursor at start of next lower line.											
Is the y-cursor outside of the page?											NO
YES											
Move y-cursor to top of page.											
Do switch on effector table entry for the character.											
ordinary	bell	back_sp	tab	new_line	line feed	car return	vert tab	form feed	char delete	line delete	default
Putchar to print the character.	Ring bell.	Decrement x-cursor by one character.	Increment x-cursor to next multiple of 8.	Move x-cursor to start of line.	Null.	Move x-cursor to start of line.	Move y-cursor up one line.	See put_ascii; put_ascii_form.	Move x-cursor back one character.	Use erase to erase entire line.	Ring bell.
Increment x-cursor by one character.				Move y-cursor down one line.					Use "erase" to erase one character.	Move x-cursor to start of line.	
Is the x-cursor outside of the page?											NO
YES											
Put cursor at start of next lower line.											
Is y-cursor outside of page?											NO
YES											
Move cursor to top of page.											

NOTE: This version of put_ascii is specific to the LSI 11 1T.

put_ascii: put_ascii_form*

Is the page bigger than or equal to the whole screen?	
YES	NO
Screen_clear.	Is the page within one character of the whole screen?
	YES NO
Screen_clear.	Use area_life to erase the page.
Put the cursor at the top left of the page.	

*This is not a procedure. It is one case in put_ascii.

`put_string (buffer, count)`

Ld_page.

Ld_cs.

Write the string.

`putchar(x, y, ch)`

Get a pointer to the vectors for this character in the current character set.

Use put to write the vectors.

NOTE: This version of putchar is specific to the LSI 11 IT.

`putdot`

Write an appropriately formatted message to the Z80 panel controller.

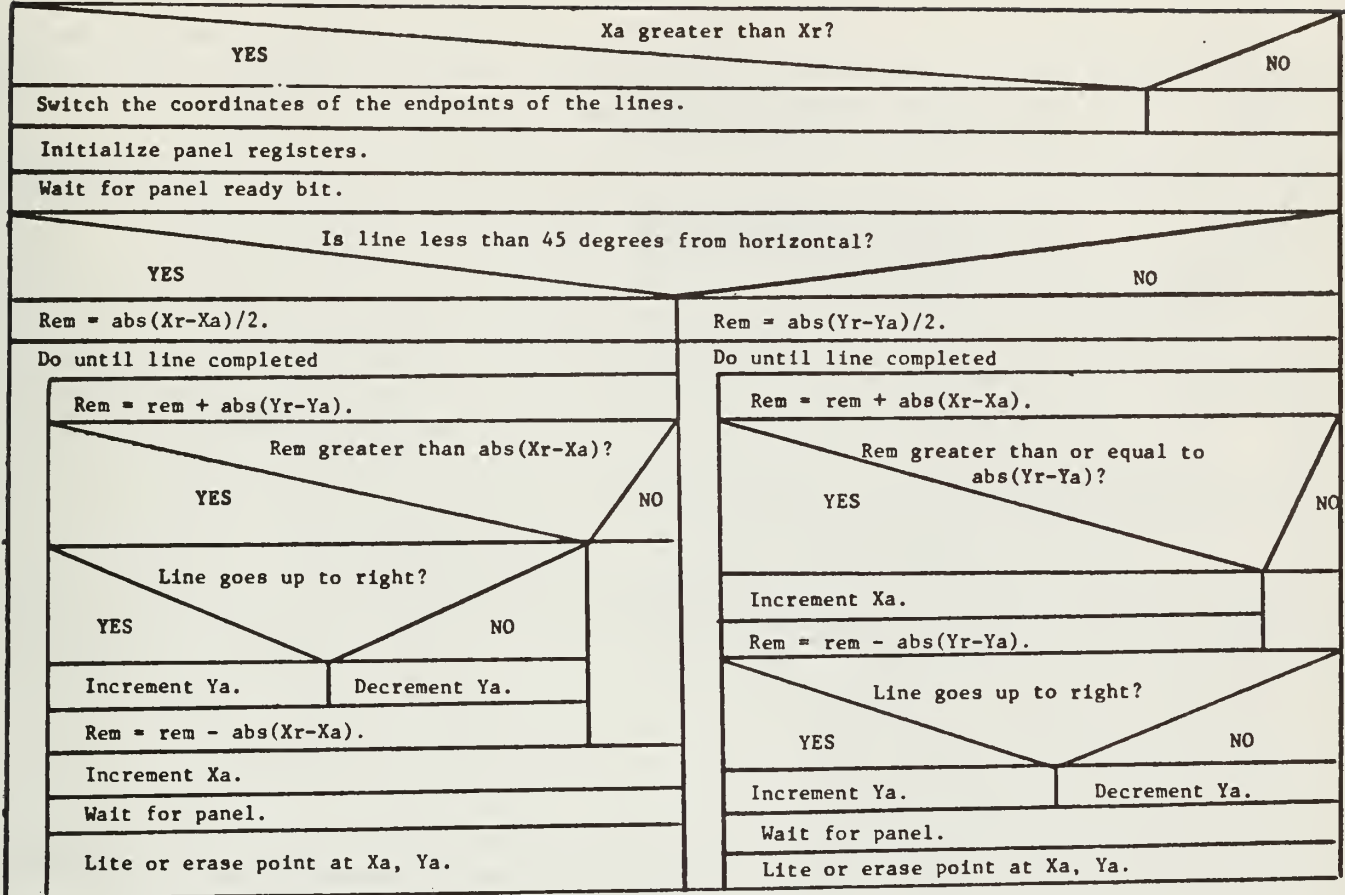
NOTE: This version of putdot is specific to the L6 IT.

`putdot (X,Y,mode)`

Set panel registers to write or erase the dot at X,Y.

NOTE: This version of putdot is specific to the LSI-11 IT.

putline(Xa, Ya, Xr, Yr, mode)



NOTE: This version of putline is specific to the LSI 11 IT.

putline

Write an appropriately formatted message to the Z80 panel controller.

NOTE: This version of putline is specific to the L6 IT.

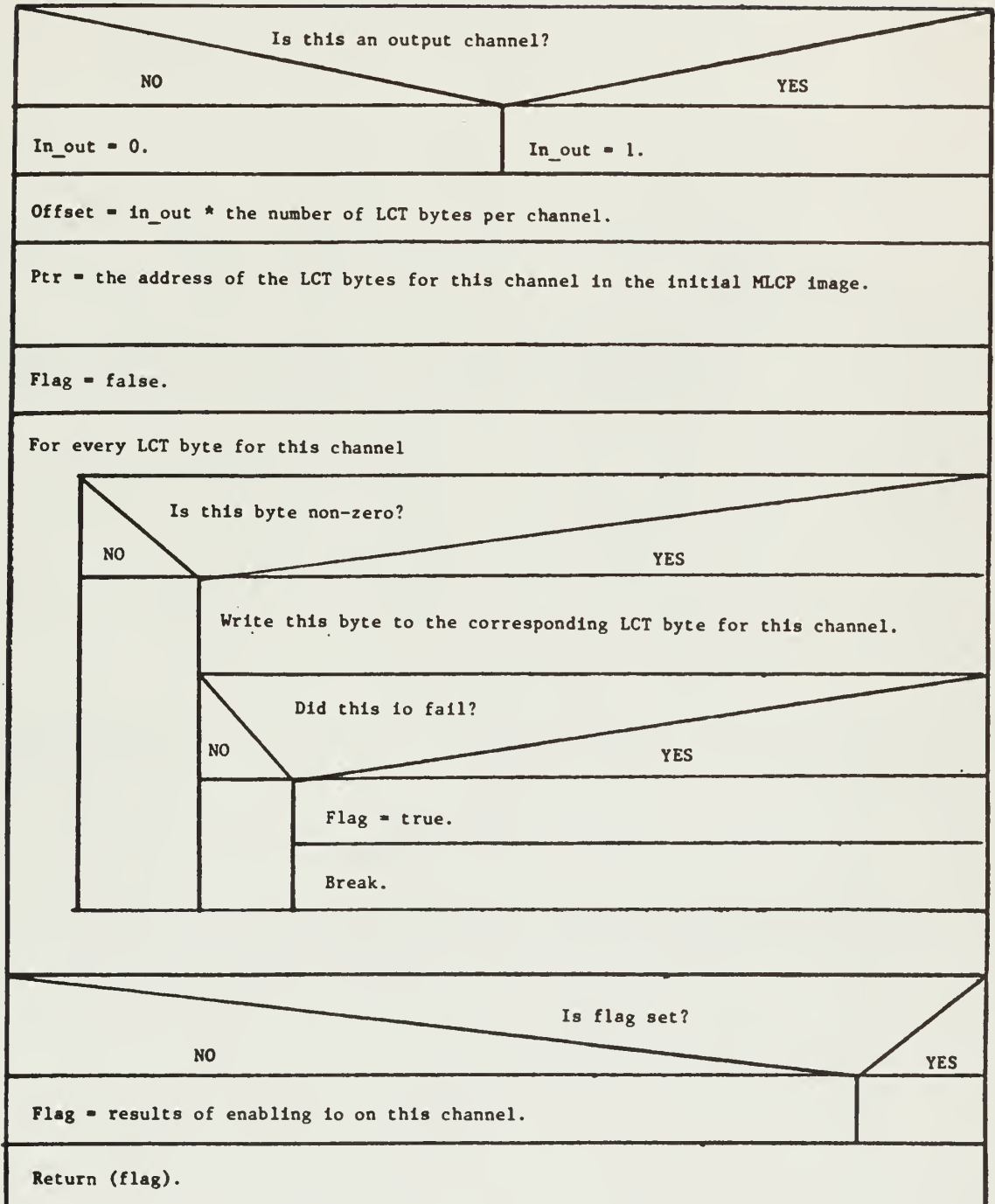
read(device_id, buf_ptr, length, &status)

Is device_id out of range?	
YES	NO
Return error.	
Does this process own the device?	
NO	YES
Return error.	
Send read request to device handler.	
Pee the request semaphore.	
Set status from that returned by handler process for read request.	
Is "catastrophic" bit on in status?	
YES	NO
Return error.	Return success.

read_q(q_ptr)

Pee the semaphore for this queue.
Deq an element from this queue.
Return the value returned by deq.


```
restart_io(chan, it_id)
```



ring_bell()

Set the panel registers to ring the bell.

NOTE: This version of ring_bell is specific to the LS111 IT.

ring_bell()

Write an appropriately formatted message to the Z80 panel controller.

NOTE: This version of ring_bell is specific to the L6 IT.

Rls_pnl()

This process owns the panels?		
YES	NO	
Decrement pp_res_level.		
pp_res_level == 0?		
YES	NO	
Set pp_owner = 0.		
Vee (pp_access).		
Return (0).		

rsrv_pnl()

This process owns the panels?		
NO	YES	
pee (pp_access).		
pp_owner = ME.		
Increment pp_res_level.		

s_read(drive, sector, buffer)

Is the sector number legal?		YES
NO		
Return error.		
Is the drive number legal?		YES
NO		
Return error.		
Compute bus address of disk.		
Compute physical track and sector number for the sector.		
Do up the maximum number of retries		
Is this first time thru for this invocation?		YES
NO		
Do a recalibrate.		
Loop forever		
Try to do an IOLD to load the buffer address and size into the interface.		
Did it work?		NO
YES		
Break.		
Have we exceeded the maximum number of attempts?		NO
YES		
Return error.		
Write track and sector numbers to the interface.		
Start a seek.		
Do until status is read		
Try to read disk status (success implies that the seek is done).		
Was there a bad error?		NO
YES		
Continue.		
Start the read.		
Do until status is read		
Try to read disk status (success implies that read is done).		
Was there an error?		YES
NO		
Return success.		
Return error.		

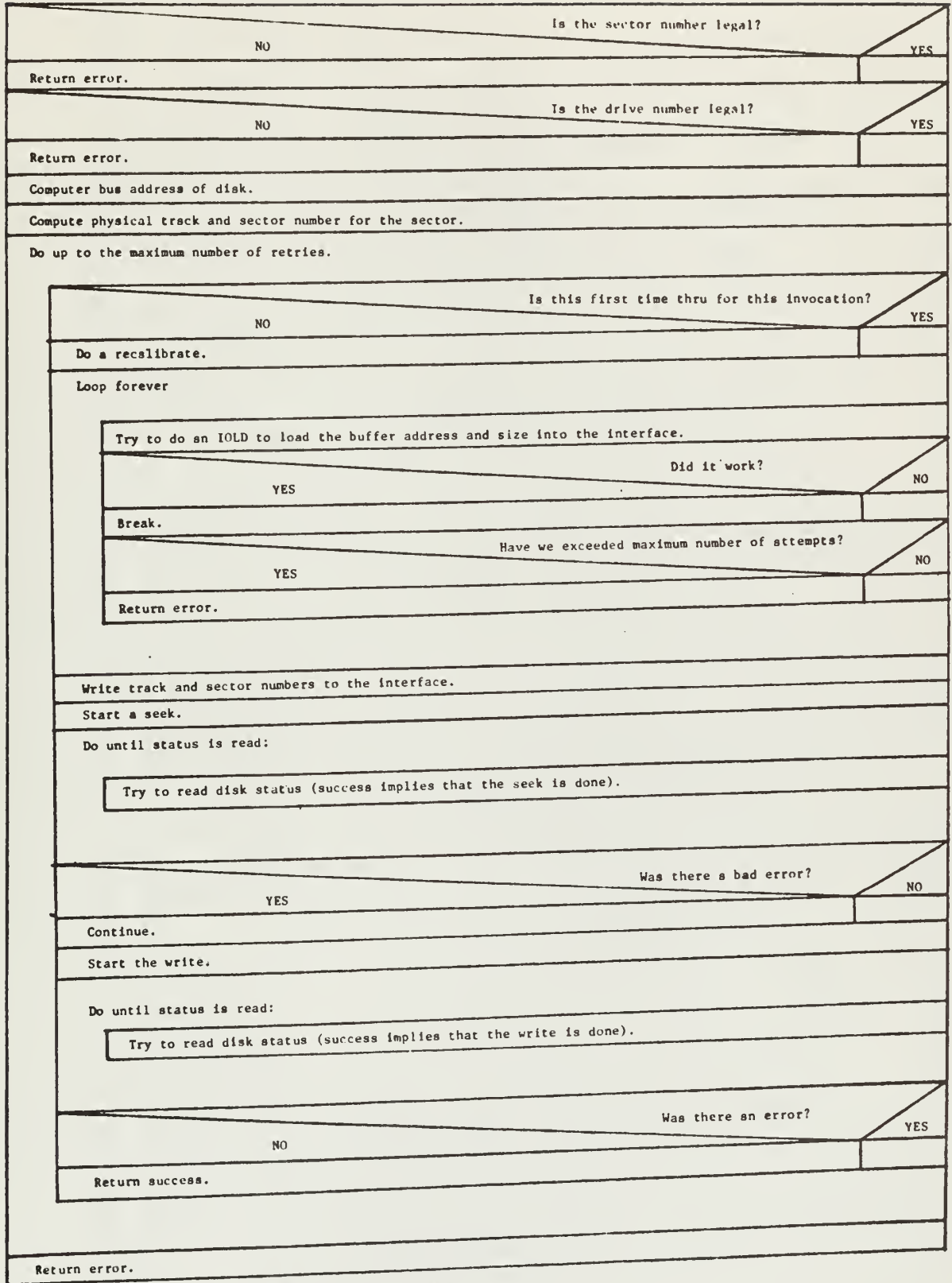
NOTE: This version of s_read is specific to the Level 6 IT.

* read(drive, sector, buffer)

Is sector or drive out of range?	
NO	YES
Return (-1).	
Drv = drive number less than 11.	
Trk = number of track containing sector.	
Sect = number of sector on track.	
Are we mapping sectors to improve access?	
NO	YES
Sect = mapped value of sect.	
For the maximum number of attempts:	
Was there an error on the disk?	
NO	YES
Reset the disk.	
Wait for the disk to be ready.	
Was there an error on the disk?	
NO	YES
Continue.	
Send drv and sect numbers to disk.	
Wait for the disk to be ready.	
Was there an error on the disk?	
NO	YES
Continue.	
Is the disk operational?	
NO	YES
Return (-1).	
Set disk's count.	
Set disk's address.	
Send track number and read command to disk.	
Wait for the disk to be ready.	
Was there an error on the disk?	
NO	YES
Continue.	
Swib (the data read).	
Return (0).	
Return (-1).	

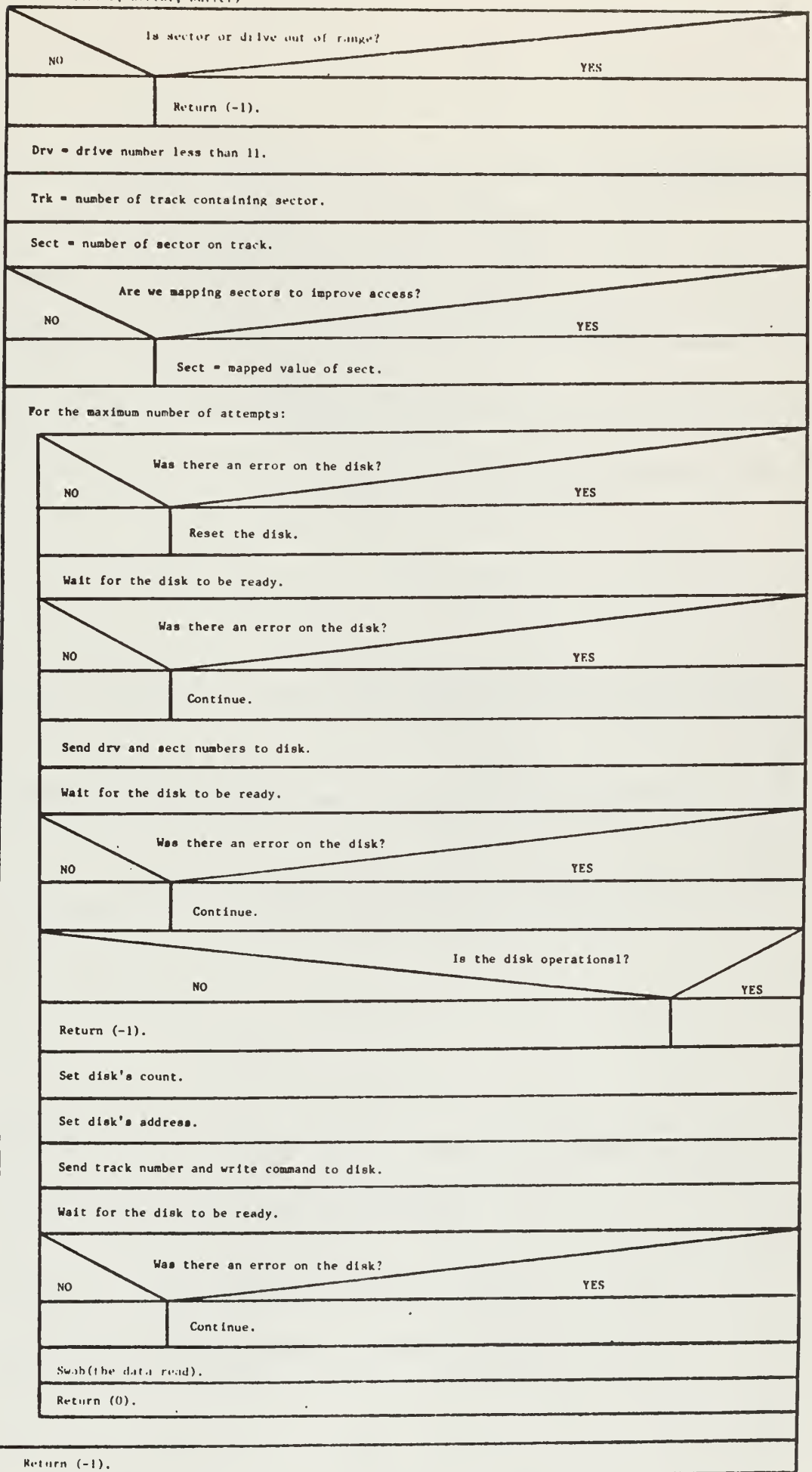
NOTE: This version of a read is specific to the ISI 11 11.

`s_write(drive, sector, buffer)`



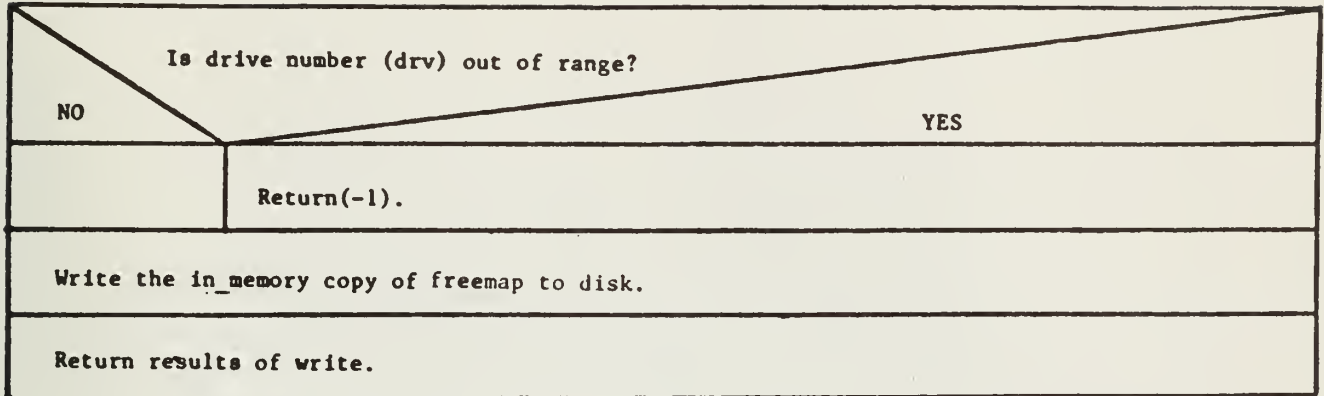
NOTE: This version of `s_write` is specific to the Level 6 IT.

• write(drive, sector, buffer)

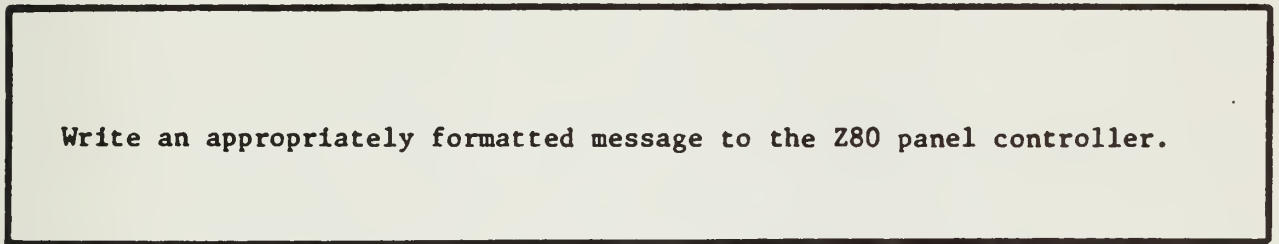


NOTE: Data direction of write is specified to the FSI 11 11.

save_free(drv)

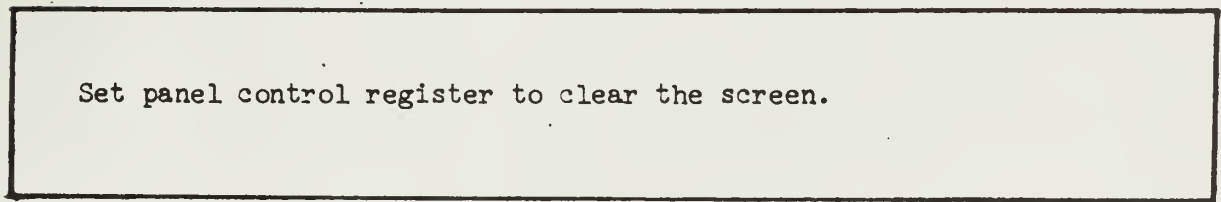


screen_clear



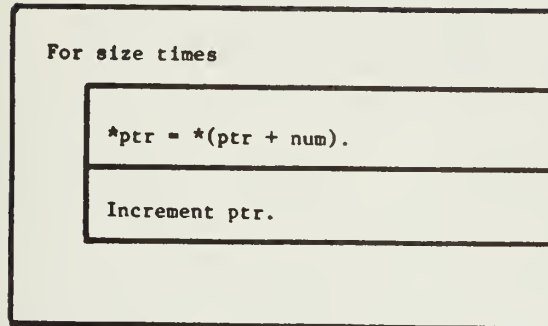
NOTE: This version of screen_clear is specific to the L6 IT.

screen_clear()

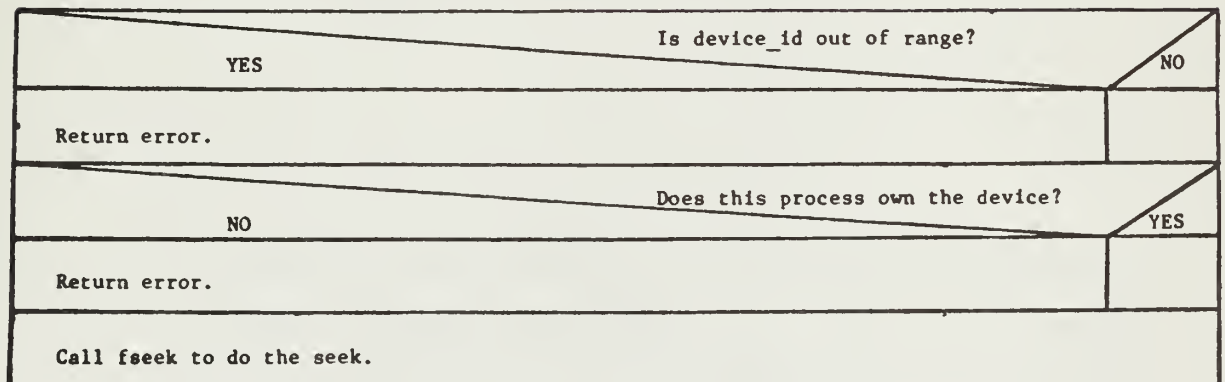


NOTE: This version of screen_clear is specific to the LSI-11 IT.

scrunch(ptr, size, num)



seek(device_id, length, type)



`set_charset(cs)`

`CS_ID = CS`

`set_cursor(x,y)`

Convert x and y from character to dot coordinates.

Assign values to `curs_x` and `curs_y`.

NOTE: This version of `set_cursor` is specific to the LSI-11 IT.

`set_cursor (x_addr, y_addr)`

Convert parameters to dot offsets.

Write offsets to display head controller.

NOTE: This version of `set_cursor` is specific to the Level 6 IT.

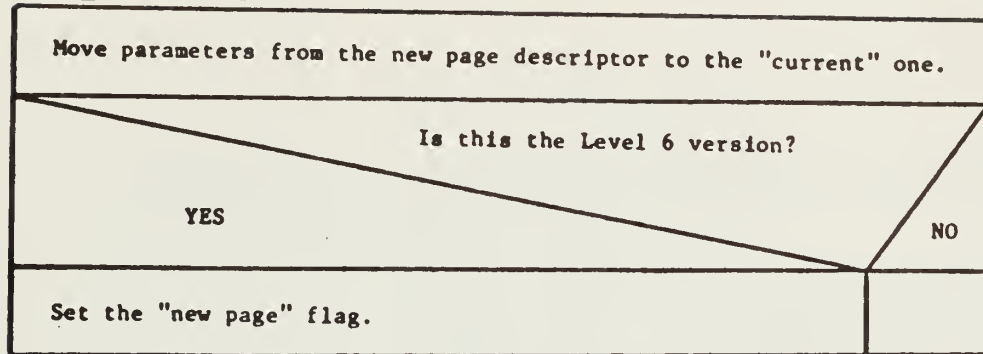
set_s.v (env_ptr)

Set_charset.	
Set_cursor.	
Set_page.	
Is this the Level 6 version?	
YES	NO
Set_pnl.	

set_mode(device_id, buf_ptr, length, &status)

Is device_id out of range?	
YES	NO
Return error.	
Does this process own the device?	
NO	YES
Return error.	
Send set_mode request to device handler.	
Pee the request semaphore.	
Set status from that returned by handler process for set_mode request.	
Is "catastrophic" bit on in status?	
YES	NO
Return error.	Return success.

set_page (page_ptr)



startup()

Size = maximum address of available memory.
Zero memory from the end of the program to the end of memory.
Set up CORETAB to indicate all of free memory.
Clear the panel.
Allocate space for the free queue elements.
Initialize the list of free queue elements.
Set the default page, character set, and cursor position to use for plasma panel printing.
Set up the READY_Q as empty.
Create all the processes specified in PROCTAB.
Initialize the I/O system.
Call first_block.

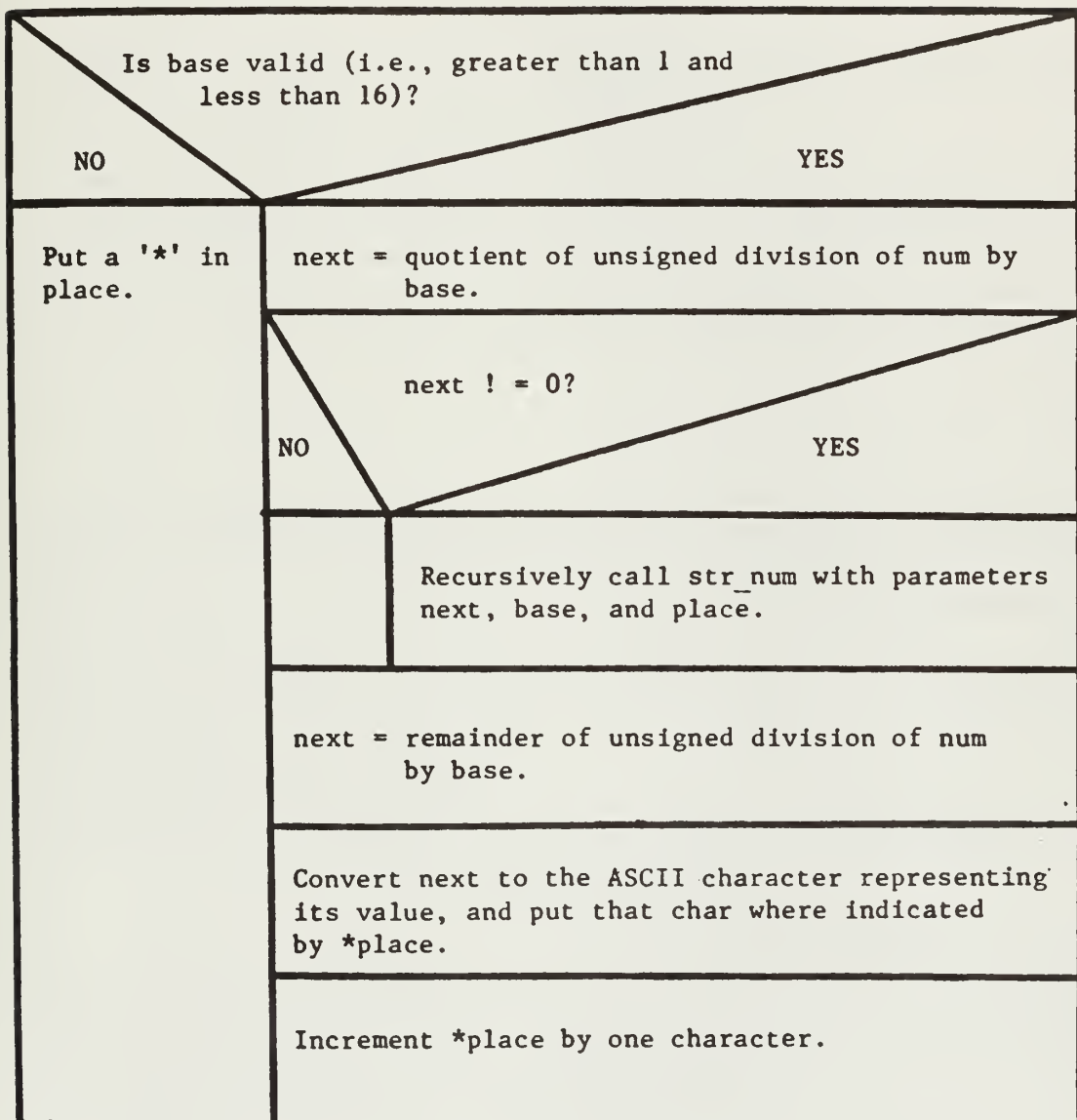
NOTE: This version of startup is specific to the LSIII IT.

startup (high)

Set up CORETAB to indicate all of memory.
Allocate space for the free queue.
Initialize the list of free queue elements.
Disable interrupts from all of the remote display heads.
Set up READY_Q as empty.
Initialize the remote display heads.
Set default page, character set, and sursor position for plasma panel printing.
Create all processes specified in PROCTAB.
Load the MLCP.
Initialize the I/O system.
Enable interrupts from all remote display heads.
Call first_block.

NOTE: This version of startup is specific to the Level 6 IT.

`str_num(num, base, place)`



`suicide()`

Clean up any pending I/O associated with this process.
Return the process' stack to the pool of free memory.
Do a process switch.

`tioid(token, separator)`

Get_cursor.
Get_page_size.
Compute length of token string.
Compute length of separator string.
Is (length of token string) + (length of separator string) more than the space left on this line?
YES
NO
Print a newline using printf.
Print the token and separator strings using printf.


```
tok_print(delim, separator, text, parma)
```

Use index to find first parameter replacement character in "text."

YES

Was one found?

NO

Change the parameter replacement character to a null.

Set "move" flag to true.

Set "move" flag to false.

Do while there are still tokens in text string:

Use get_token to get one token from text.

YES

Was one found?

NO

Increment "text" pointer to past the token.

Is "move" flag on, indicating a parm to expand?

YES

NO

Does "text" now point to a null?

YES

NO

Move "text" pointer up to point at the parameter replacement character that was nulled out.

Change the null back to what it was.

Use parm_xpand to expand the parameter.

Increment "text" pointer past the parameter replacement stuff.

Do while there are still tokens:

Use get_token to get one token from the expanded parameter.

YES

Did we get a token?

NO

Point past the token.

Are we at the end of the expanded parameter and is the character following the parameter a delimiter?

YES

NO

Use tioid to print the token.

Use tioid to print the token followed by a separator.

Use index to find the next parameter replacement character.

YES

Was one found?

NO

Change the character to null.

Set the "move" flag off.

Set the "move" flag on.

Set flag so outer loop continues.

tp_driver()

Do forever

Read a word from input queue.

YES

Is the word odd?

NO

Request code = "data."

The word is a pointer to a request block containing the request code.

The touch coordinates are in bits 2 - 14 of the word.

Switch on request code.

open

Set "open" flag on.

close

Set "open" flag off.

flush

Empty the buffer.

peek

Put the count of characters in buffer into request block.

data

(See tp_driver: data_type)

read

(See tp_driver: read_type)

default

Error:
Reject the request.

Empty the input buffer.

Disable touch panel input.

Is a read outstanding?

YES

Enable touch panel input.

See user's semaphore.

Set data length to zero and reject the saved request.

NO

See user's semaphore.

See user's semaphore.

NOTE: This version of tp_driver is specific to the LSI 11 IT.

tp_driver: data_type*

Is the touch panel open?		YES	NO
Is there a read saved up?		YES	NO
Copy the touch into the user's buffer.	Is there space in buffer?	YES	NO
Vee user's semaphore.			
Reset "read pending" flag.	Copy the touch into the internal buffer.		

*This is not a procedure. It is one case in tp_driver.

tp_driver: read_type*

Is there data available?		YES	NO
Compute number of touches to give to user.	Save the read request.		
Copy touches into user's buffer.			
Vee user's semaphore.			

*This is not a procedure. It is one case in tp_driver.

tp_driver

Do forever:

Read a word from input queue.

Is it a known special code?

YES

NO

The word is the request code.

The word is a pointer to a request block containing the request code.

Switch on request code.

open	close	flush	peek	overflow or data	read	setmode type	default
Set "open" flag on.	Set "open" flag off.	Empty the buffer.	Put count of characters in buffer into request block.	Is a read outstanding?		Is this a set panel aetmode?	Error: reject the request.
Empty the input buffer.	Disable touch panel input from all panels.	Is a read outstanding?	Put count of characters in buffer into request block.	NO		YES	
Enable touch panel input from all panels.	Vee user's semaphore.	YES	Vee user's semaphore.	Break.		Enable input from all panels specified.	Error: reject the request.
		Set data length to zero and reject the saved request.		Use the request block from the previous read.		Disable input from all panels not specified	
				Is data available?		Vee user's semaphore.	
				YES			
				NO			
				Copy data into user's buffer.	Save pointer to the request block.		
				Vee user's semaphore.			

NOTE: This version of tp_driver is specific to the Level 6 IT.

tt_activate(t)

For every slot in tt_current

Is this slot empty?

NO

YES

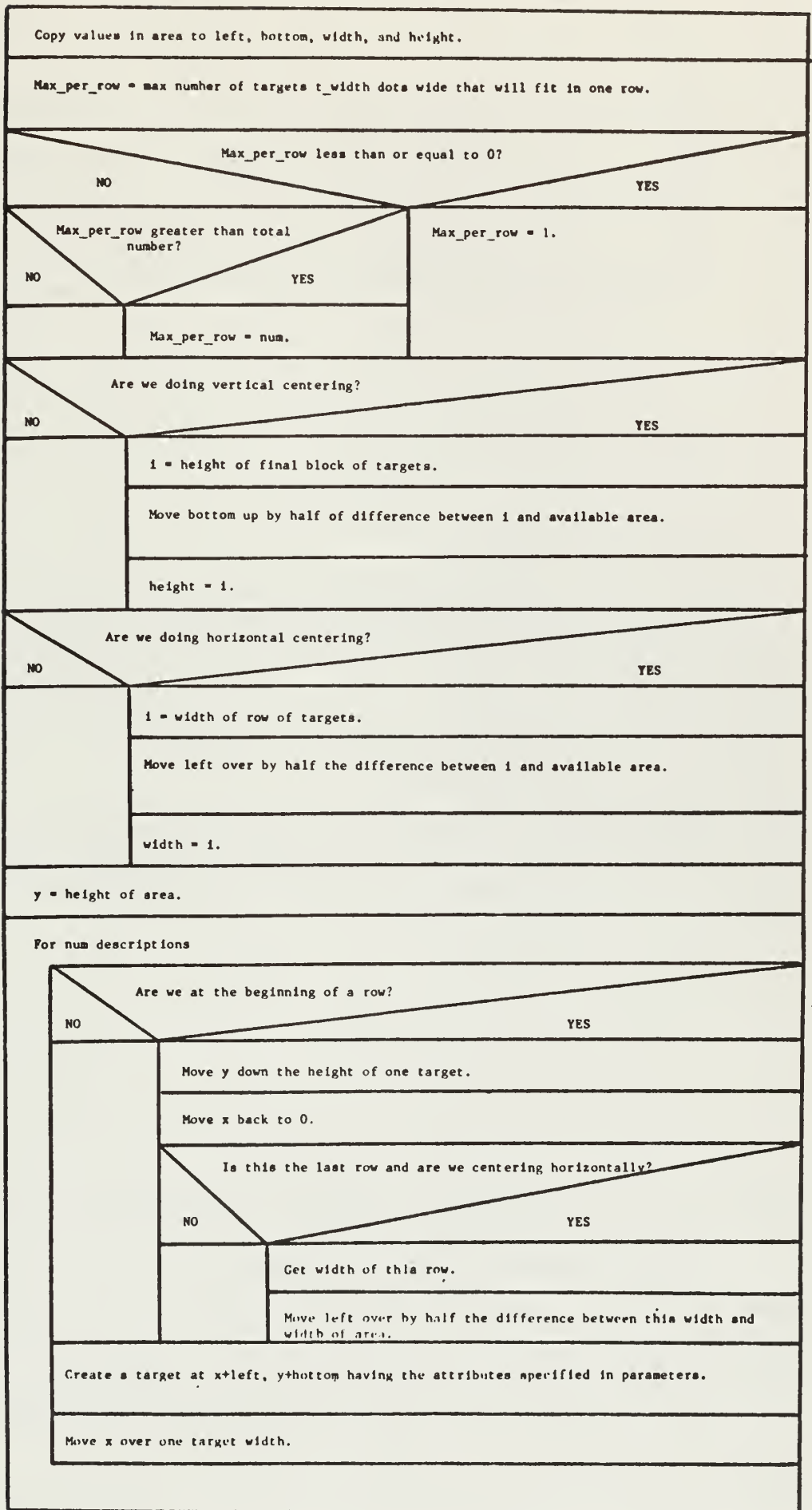
tt_current(slot) = t.

Display the target.

Return(slot).

Return (-1).

tt_arranger(list, num, values, labels, va, flags, area, t_width, t_height, mode)



tt_cleanup()

For every slot in tt_current

Delete the target pointed to by the entry in that slot.

tt_create(t, x, y, width, height, value, label, cs, flag)

Copy the parameters into the appropriate fields of the target structure.

Reset the in_use flag.

Is tflags set so that the target would appear on no remote display heads?

YES

NO

Set tflags so the target will appear on head 0, only.

NOTE: This version of tt_create is specific to the Level 6 IT.

tt_create(t, x, y, width, height, value, label, cs, flag)

Copy the parameters into the appropriate fields of the target structure.

Reset the in_use flag.

NOTE: This version of tt_create is specific to the LSI-11 IT.

tt_deactivate(slot)

Erase the target identified by slot.

Delete the target.

Return the results of erasing and deleting.

tt_delete(slot)

Does slot indicate an active target?	
NO	YES
Return (-1).	
Turn off in_use flag in target indicated by slot.	
Zero tt_current(slot).	
Return (0).	

tt_flash(slot)

Does slot indicate an active target?	
NO	YES
Return (-1).	
Is this target flashable?	
NO	YES
	Lite all the dots in the target.
	Erase them all.
	Display the target.
Return (0).	

tt_label(alot)		Does alot indicate an active target?	YES
NO			
Return (-1).			
t = pointer to target structure for target.			
Is this a target that should not be labeled?			
NO			YES
Return (0).			
Rerv_pnl.			
Remember current printing environment.			
Call set_pnl with bits from tflags so that the target will be displayed on appropriate panel(s).			
Set x and y to coordinates of lower left corner of target, leaving white space inside border.			
Set width and height to the size of the target, leaving white space inside the edges.			
Set the charset to the one for the target.			
Get the size of characters in this charset.			
Modify x and width to center a maximum line of characters within the target.			
Set the printing page to be this area within the target.			
Position the cursor at the top of the page.			
Tok_print the label.			
Restore the printing environment.			
Ris_pnl.			
Return (0).			

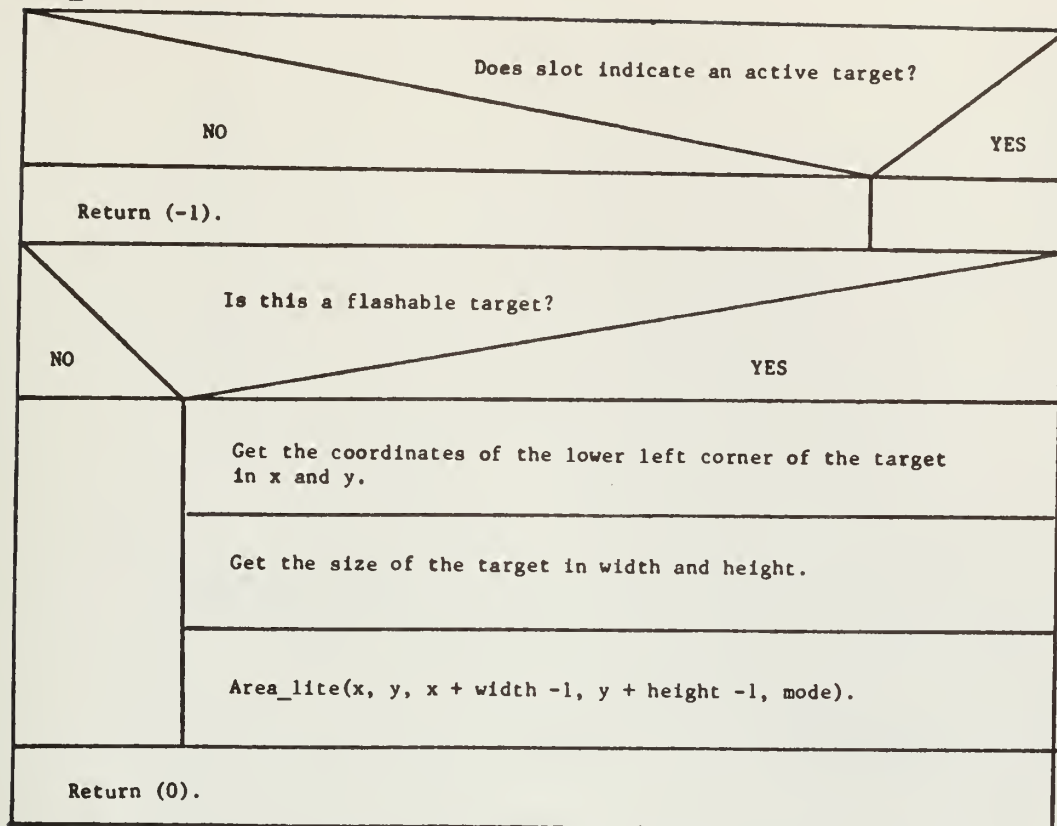
NOTE: This version of tt_label is specific to the Level 6 IT.

tt_label(slot)

Does slot indicate an active target?	
NO	YES
Return (-1).	
t = pointer to target structure for target.	
Is this a target that should not be labeled?	
NO	YES
	Return (0).
Remember current printing environment.	
Set x and y to coordinates of lower left corner of target, leaving white space inside border.	
Set width and height to the size of the target, leaving white space inside the edges.	
Set the charset to the one for the target.	
Get the size of characters in this charset.	
Modify x and width to center a maximum line of characters within the target.	
Set the printing page to be this area within the target.	
Position the cursor at the top of the page.	
Tok_print the label.	
Restore the printing environment.	
Return (0).	

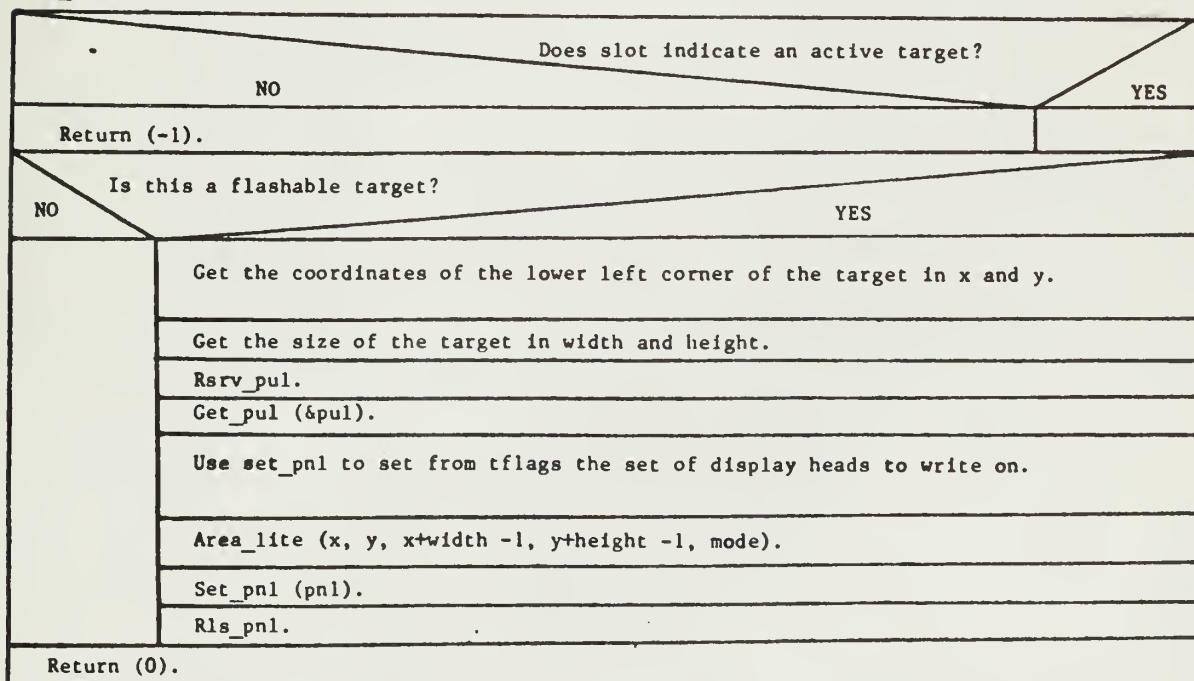
NOTE: This version of tt_label is specific to the LSI-11 IT.

tt_lite(slot, mode)



NOTE: This version of tt_lite is specific to the LSI-11 IT.

tt_lite(slot, mode)



This version of tt_lite is specific to the Level 6.

tt_mark(slot, mode)

Does slot indicate an active target?	
NO	YES
Return (-1).	
Is this a markable target?	
NO	YES
Return (0).	
Get coordinates of lower right corner of target, offset to be inside boarder, in x2, y1.	
Get coordinate of left edge of mark in x1.	
Get coordinate of top edge of mark in y2.	
Area_lite (x1, y1, x2, y2,.....).	
Return (0).	

NOTE: This version of tt_mark is specific to the LSI 11 IT.

tt_mark(slot, mode)

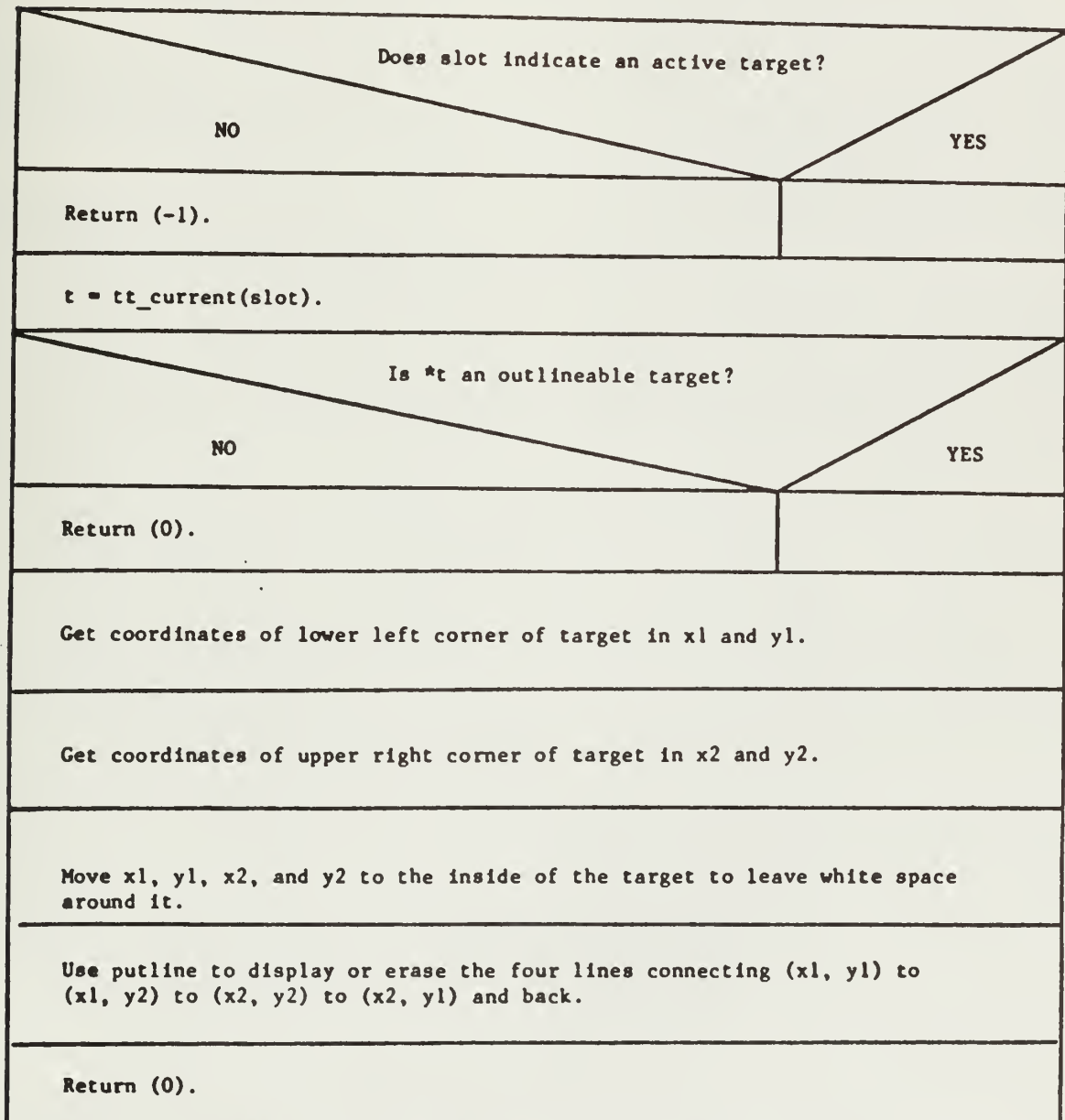
Does slot indicate an active target?	
NO	YES
Return (-1).	
Is this a markable target?	
NO	YES
Return (0).	
Get coordinates of lower right corner of target, offset to be inside border, in x2, y1.	
Get coordinate of left edge of mark in x1.	
Get coordinate of top edge of mark in y2.	
Rsrv_pnl.	
Get_pnl(&pnl).	
Use set_pnl to set from tflags the set of remote display heads to be written on.	
Area_lite(x1, y1, x2, y2, mode).	
Set_pnl(pnl).	
Return (0).	

NOTE: This version of tt_mark is specific to the Level 6 IT.


```
tt_move(t, new_x, new_y)
```

Set tx and ty entries in target structure t to new_x and new_y.

```
tt_outline(slot, mode)
```



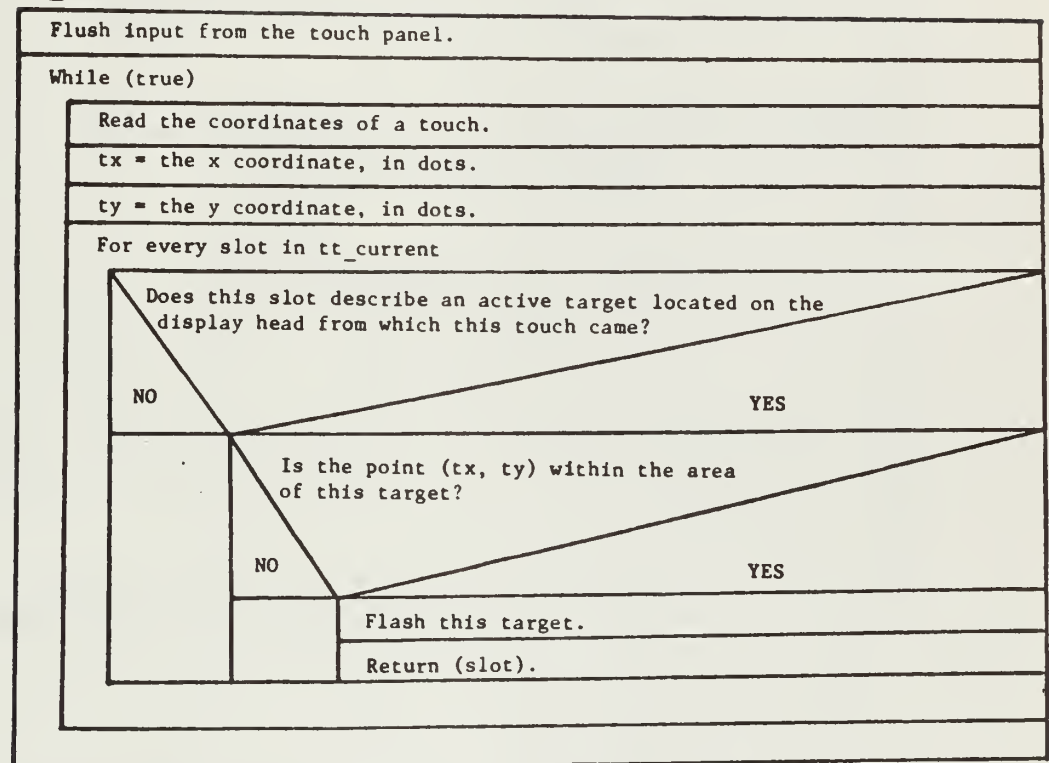
NOTE: This version of tt_outline is specific to the LSI-11 IT.

tt_outline(slot, mode)

Does slot indicate an active target?	
NO	YES
Return (-1).	
t = tt_current(slot).	
Is *t an outlineable target?	
NO	YES
Return (0).	
Get coordinates of lower left corner of target in x1 and y1.	
Get coordinates of upper right corner of target in x2 and y2.	
Move x1, x2, and y2 to the inside of the target to leave white space around it.	
Rsrv_pnl.	
Use get_pnl to save the set of currently selected remote display heads.	
Use set_pnl to set the set of selected heads to those specified in tflags.	
Use putline to display or erase the four lines connecting (x1, y1) to (x1,y2) to (x2, y2) to (x2, y1) and back.	
Use set_pnl to put the set of selected heads back.	
Rls_pnl.	
Return (0).	

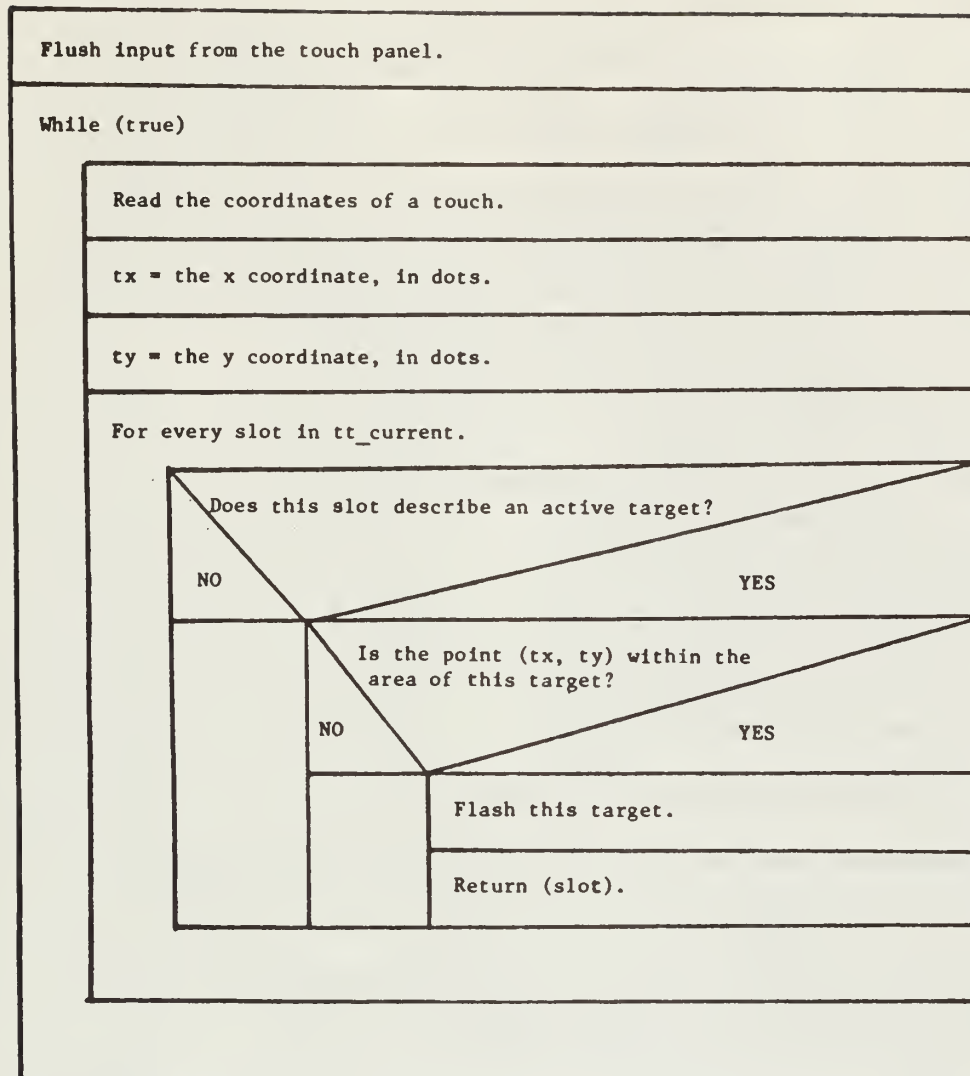
NOTE: This version of tt_outline is specific to the Level 6 IT.

tt_read(touch)



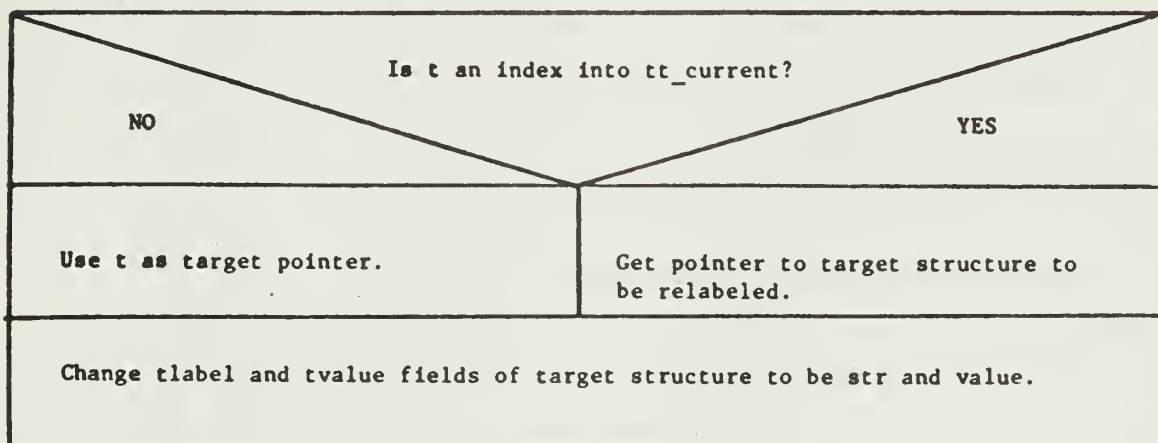
NOTE: This version of tt_read is specific to the Level 6 IT.

tt_read(touch)



NOTE: This version of tt_read is specific to the LSI 11 IT.

tt_relabel(t, str, value, mode)




```
tt_selections(touch, max num tchs, num, ovrf1, values, slots)
```

```
While (true)
```

```
Slot = results of reading one touch via tt_read.
```

```
t = tt_current(slot).
```

```
add_to_list = true.
```

```
Do duplicate touches cancel each other?
```

NO

YES

```
For every target chosen so far
```

```
Does value of chosen target == t_value?
```

NO

YES

```
Decrease count of chosen targets.
```

```
Scrunch the arrays of target slots and target values.
```

```
Un_mark the target.
```

```
add_to_list = false.
```

```
Break.
```

```
Is add_to_list true?
```

NO

YES

```
Is the number chosen greater than or equal to the  
max number allowed?
```

NO

YES

```
Should we return in this case?
```

NO

YES

```
Return (num chosen).
```

```
Decrement number chosen.
```

```
Delete oldest choice from lists.
```

```
Add target to list of choices.
```

```
Mark target.
```

```
Increment count of chosen targets.
```

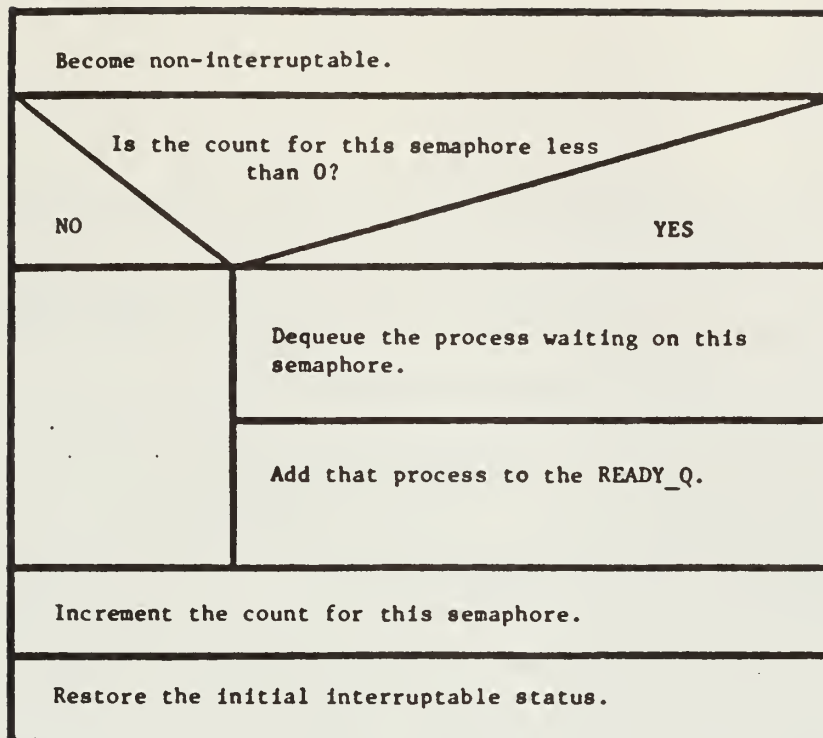
```
Does this target have the force-return bit set?
```

NO

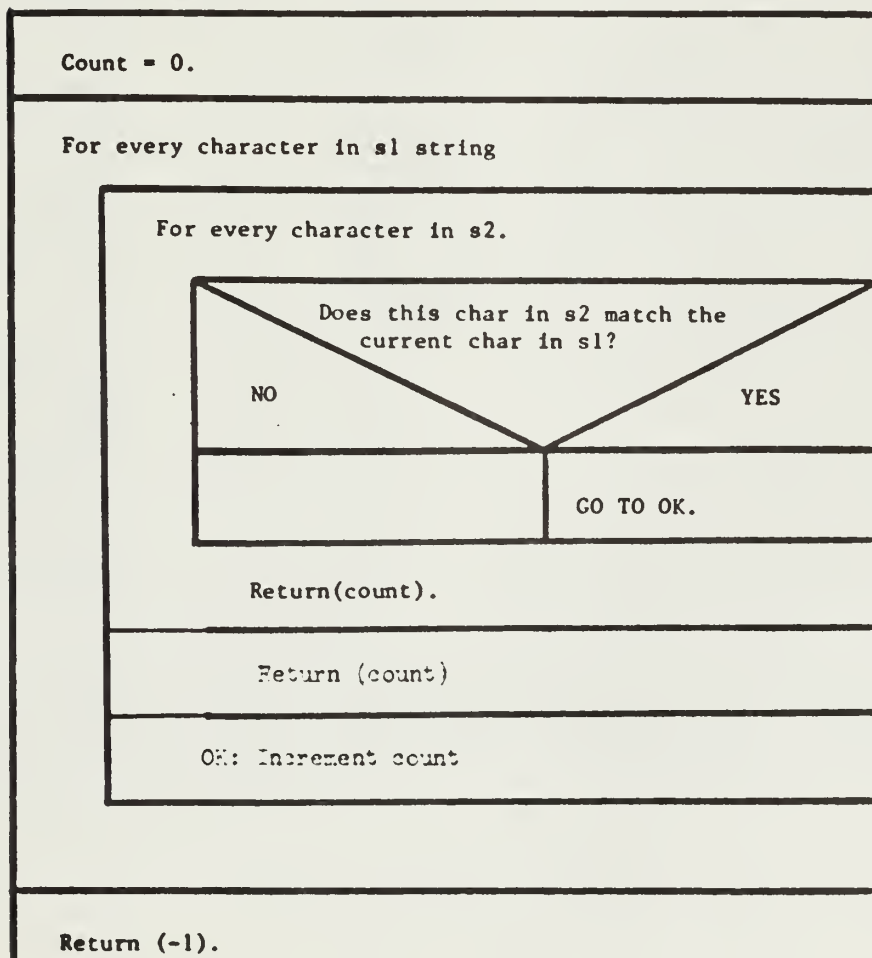
YES

```
Return (number chosen).
```


vee(sem)



verify(s1, s2)



vip_proc()

Initialize DUVII Device.

Start transmission of a quiescent frame.

Do forever

Read word from input queue.

Is word a small odd integer?

YES

NO

The word is the request code.

The word is a pointer to a request block with the request code.

Switch on the request code.

open	close	flush	vip up	vip down	data	read	write	ack	peek	default
Zero saved read and write indicators.	Set "open" flag off.	Null.	Set "vip_ready" flag on.	Set "vip_ready" flag off.	Is a read pending?	Null.	Is a write pending?	Null.	Put count of characters in input buffer into the request block.	Reject the request.
Vip line operating?	Empty input buffer.						YES	NO		
NO	YES	Outstanding read?			Break.		Reject the new write request.			
Reject request.	Set "open" flag on.	YES			Get pointer to request block for the pending read.		Break.			
		Reject outstanding request.					Set this up as the current write request.			
					See vip_proc: vip_read		See vip_proc: vip_write			
Vee user's semaphore.	Outstanding write?									
	YES									
	Reject outstanding request.									
	Vee user's semaphore.									

NOTE: This version of vip_proc is specific to the LSI 11 IT.

vip_proc: vip_read*

Is this a read request and another read is already pending?	
YES	NO
Reject the new read request.	
Break.	
Is the buffer empty?	
YES	NO
Save a pointer to this request block as a pending read.	
Break.	
While there are still input characters and the user's buffer is not exhausted.	
Copy a character into the user's buffer.	
Was the character an ETX?	
YES	NO
Break.	
Null terminate the user's buffer.	
Vee the user's semaphore.	

*This is not a procedure. It is one case in vip_proc.

vip_proc: vip_write*

Is a write active?		YES
NO	Break.	
Has all data for this write been written?		NO
YES	Vee user's request.	
	Set flag indicating no write pending.	
	Break.	
Construct an output message, using as much data from the current write request as will fit in the output buffer.		
Set buffer pointer and count for the output interrupt handler.		
Set flag for input interrupt processor indicating that an output message is available.		

*This is not a procedure. It is one case in vip_proc.

`vip_proc()`

Empty input buffer.									
Set flag indicating that the line is down and closed.									
Do "forever"									
Read a word from VIP_Q.									
Is it a known special code?									
YES									
NO									
The word is a request code.									
The word is a pointer to a request block containing a request code.									
Switch on the request code.									
open_type	vip_down	close_type	flush_type	vip_up	data	read_type	write_type	ack	peek
See vip_proc::open_type	Set line up flag to false.	Flag line as closed.	See vip_proc::flush_type	Set line up flag to true.	Is a read saved?	YES	Is a write saved?	YES	Put count of characters in input buffer into user's request block.
See vip_proc::open_type	Turn off input and output.	Turn-off successful?	See vip_proc::flush_type	Make it look as though the read just came in.	NO	Break.	Reject the new request.	Break.	Vee the request.
See vip_proc::open_type	Set error flags for user routine.	NO	YES	See vip_proc: read_type.	Init count and address for current write.	See vip_proc: read_type.	Init count and address for current write.	See vip_proc: ack.	See vip_proc: ack.

NOTE: This version of vip_proc is specific to the Level 6 IT.

vip_proc: ack

Is there s current write?	
NO	YES
Break.	
Is the count zero (i.e. is the write now completed)?	
YES	NO
Set return to user to show successful write.	
Vee user's request.	
Break.	
Format an output message, using as much of the user's data as will fit in the buffer.	
Decrease current count and incresse current address by the number of user's characters put in messsge.	
Set a flag so that the interrupt routine will initiate transmission of this message when the time comes.	

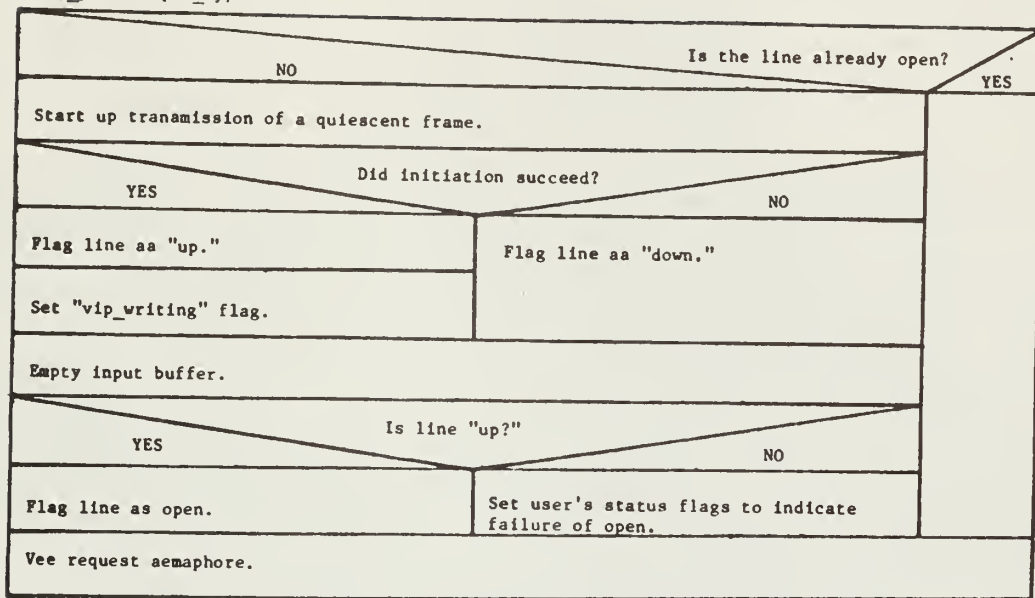
*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.

vip_proc: flush_type*

Is this s close or a flush?	
YES	NO
Empty the input buffer.	
Is s read saved up?	
YES	NO
Reject the saved read.	
Is a write saved up?	
YES	NO
Reject the write.	
Is this a close or s flush?	
YES	NO
Vee the request semaphore.	

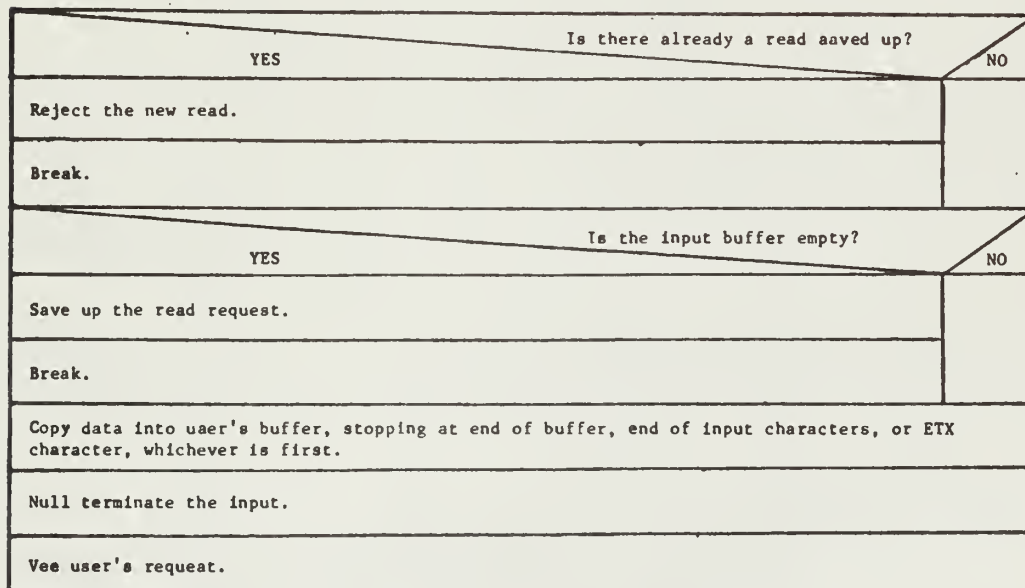
*This is not a procedure. It is one csse in vip_proc and is specific to the Level 6 IT.

vip_proc: open_type*



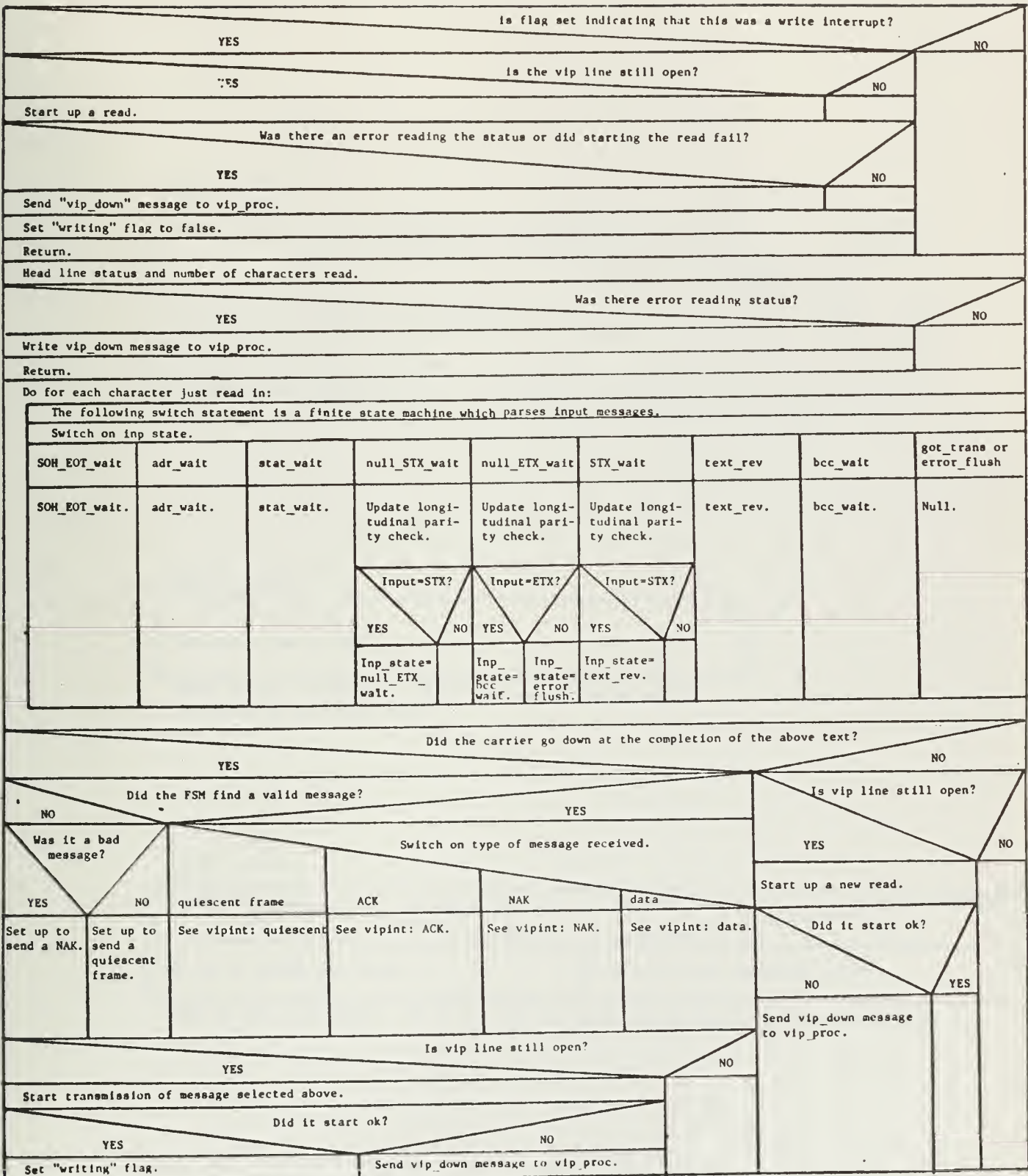
*This is not a procedure. It is one case in vip_proc and is specific to the Level 6 IT.

vip_proc: read_type*



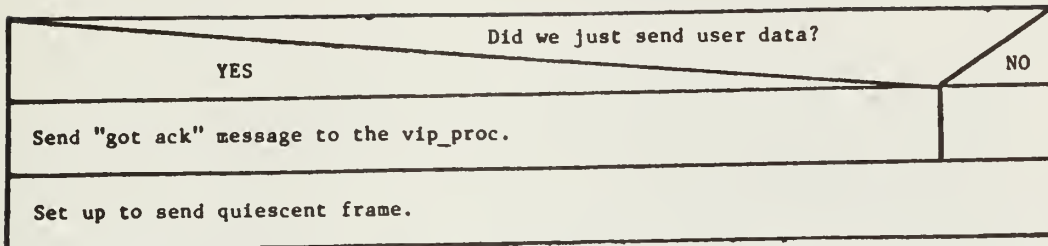
*This is not a procedure. It is one caase in vip_proc and is specific to the Level 6 IT.

vipint()



NOTE: This version of vipint is specific to the Level 6 IT.

vipint: ACK*



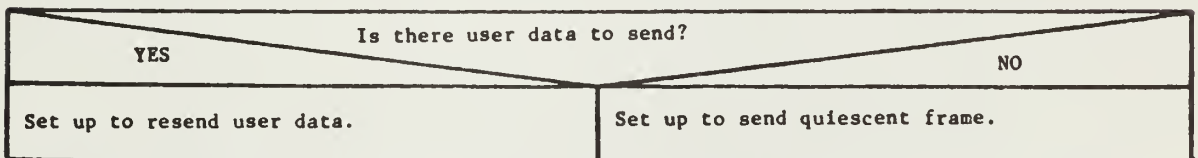
*This is not a procedure. It is one case in vipint and specific to the Level 6 IT.

vipint: data*



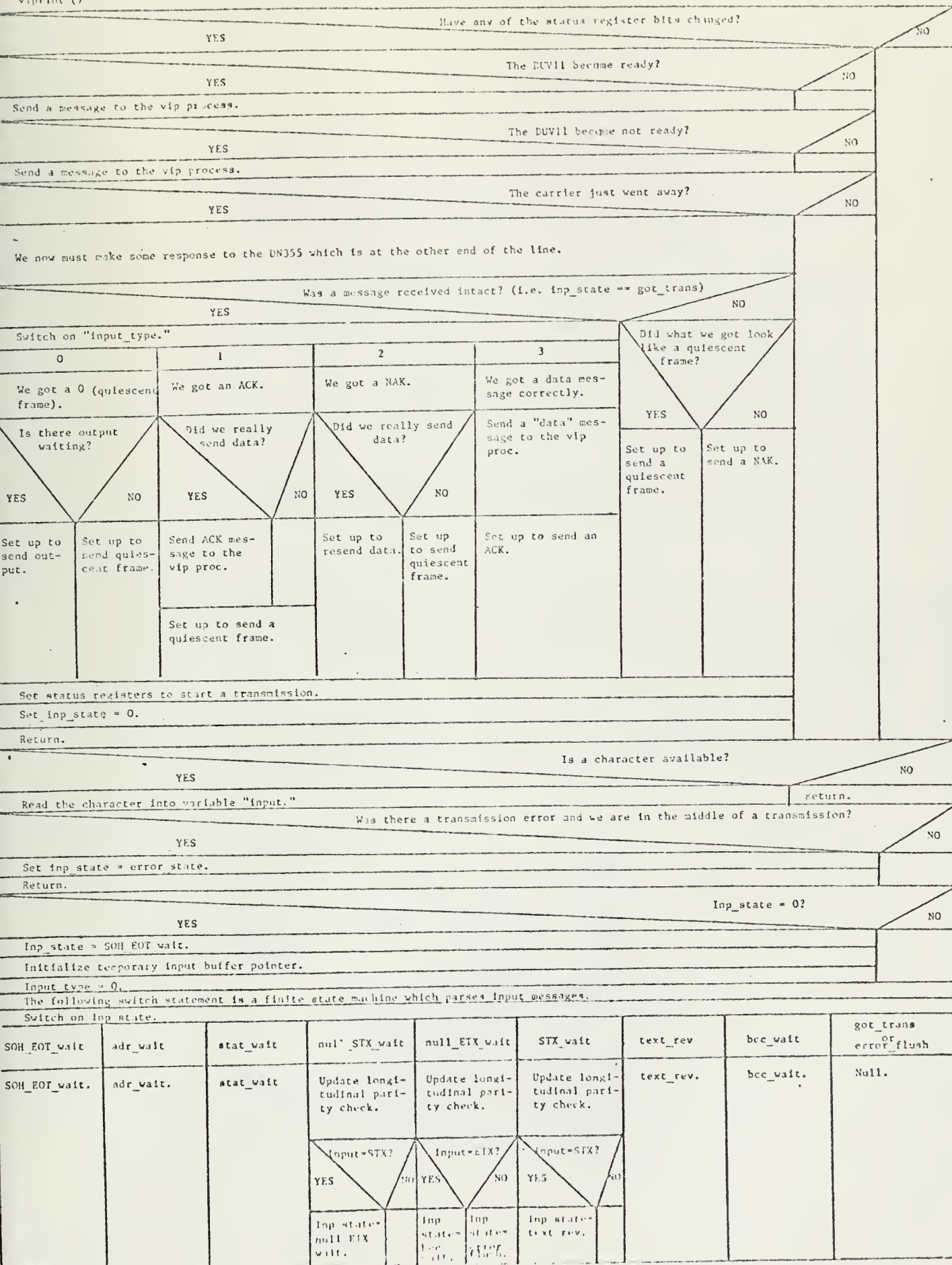
*This is not a procedure. It is one case in vipint and is specific to the Level 6 IT.

vipint: NAK*



*This is not a procedure. It is one case in vipint and is specific to the Level 6 IT.

viprint ()



adr_wait*

Update longitudinal parity check.		
Input = correct address?		
YES	NO	
Inp_state = stat_wait.	Input = EOT?	
	YES	NO
	Inp_state = got_trans. i.e. transmission received ok.	Inp_state = error_flush. i.e. transmission received in error.

*This is not a procedure. It is one case in viprint.

bcc_wait*

Is the input character equal to the longitudinal parity check character?	
NO	YES
Inp_state = error_flush.	Inp_state = SOH_EOT_wait.

*This is not a procedure. It is one case in viprint.

SOH_EOT_wait*

Input = SOH?		
YES	NO	
Inp_state = adr_wait.	Input = EOT?	
	YES	NO
	Inp_state = got_trans. i.e. transmission received ok.	
Initialize longitudinal parity check character.		

*This is not a procedure. It is one case in viprint.

stat_wait*

Update longitudinal parity check.			
Input = null?			
YES		NO	
Input_type = 3.	Input = ACK?		
1.e. this is an input text message.	YES		NO
	Input_type = 1.	Input = NAK?	
Inp_state = STX_wait.	1.e. this is an ACK message.		NO
	Inp_state = null_STX_wait.	Input_type = 2.	Inp_state = error_flush.
		1.e. this is a NAK message.	
		Inp_state = null_STX_wait.	Transmission is in error.

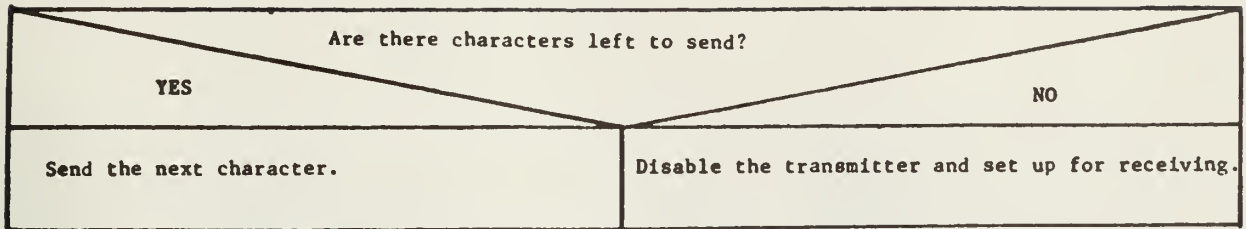
*This is not a procedure. It is one case in viprint.

text_rev *

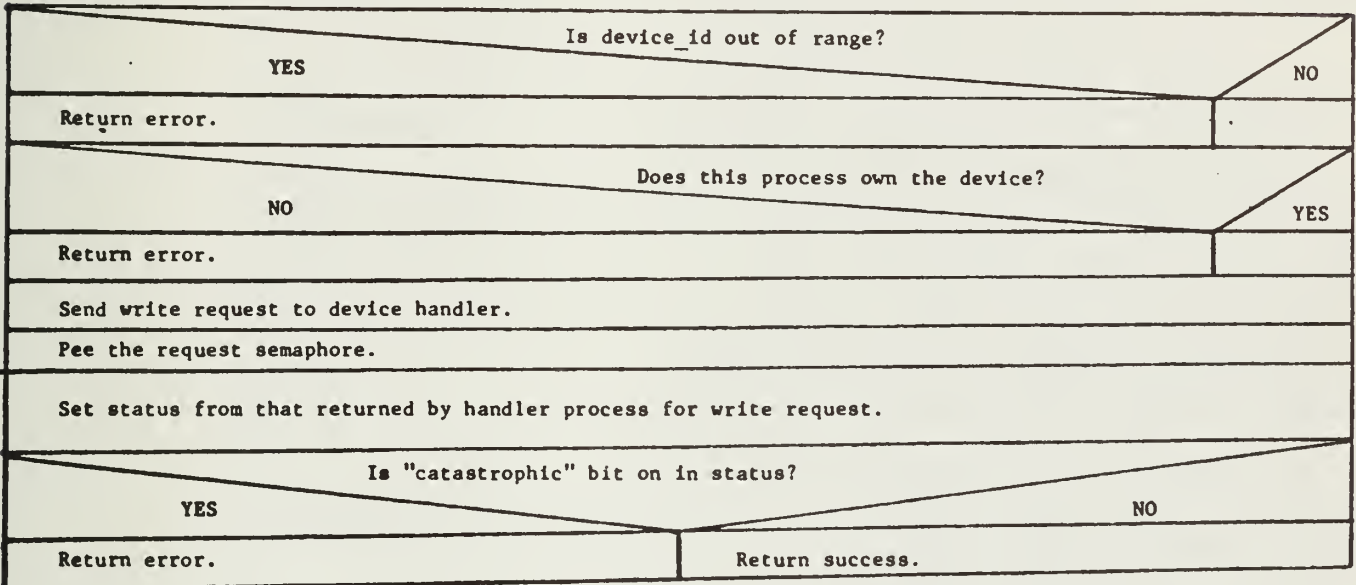
Update longitudinal parity check.		
Would this character overflow the buffer?		
YES		
NO		
Inp_state = error_flush.	Insert the character into the input buffer.	
	Was it an ETX?	
	NO	YES
	Inp_state = bcc_wait.	

*This is not a procedure. It is one case in viprint.

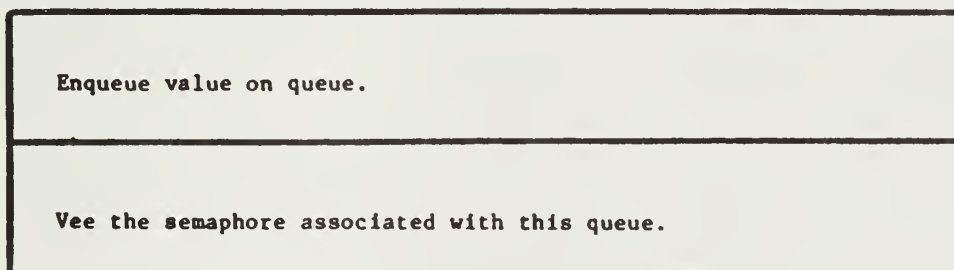
`vipxint()`



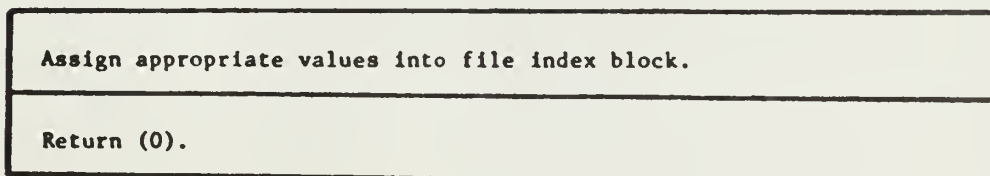
`write(device_id, &status)`



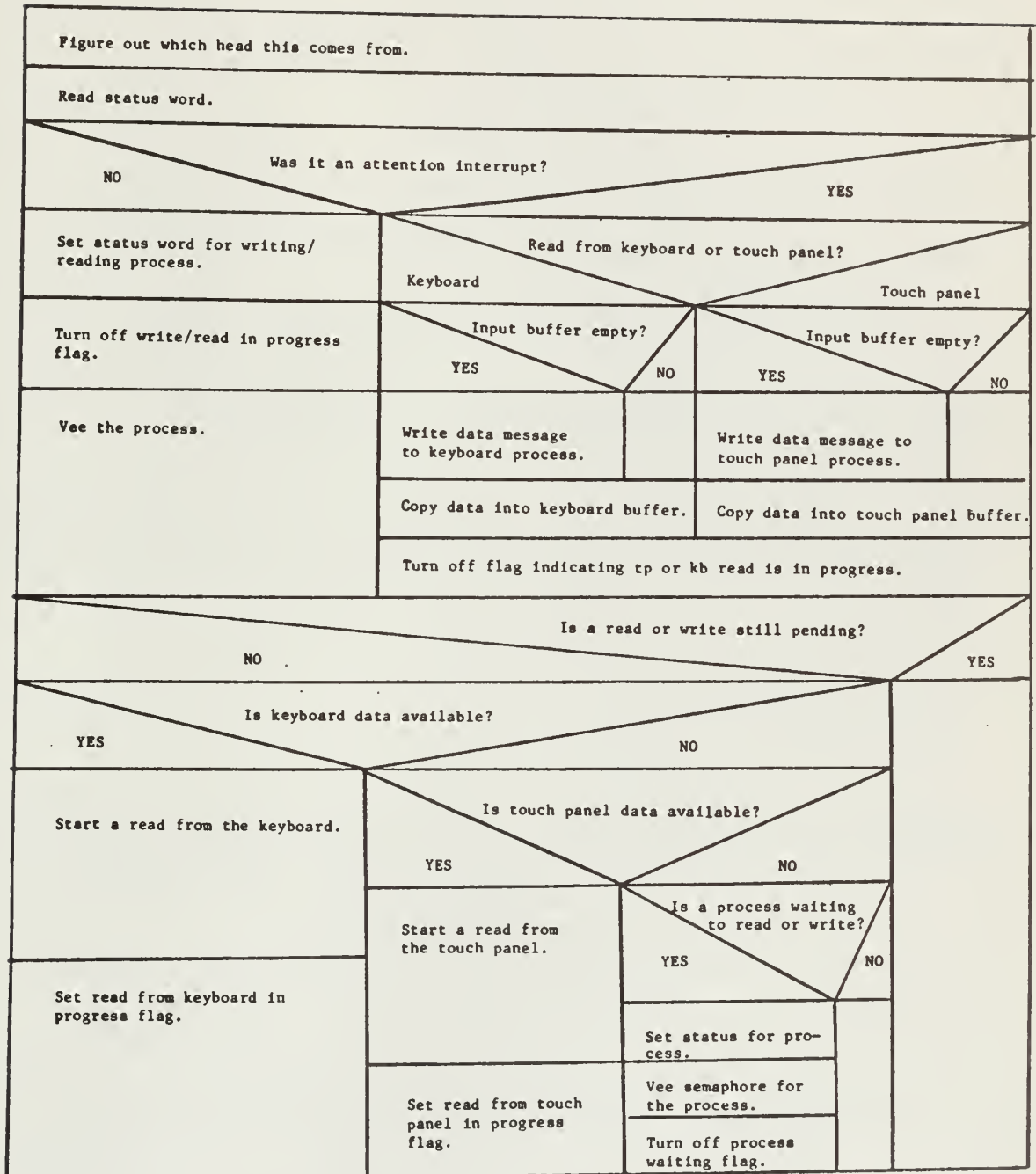
`write_q(q_ptr, value)`



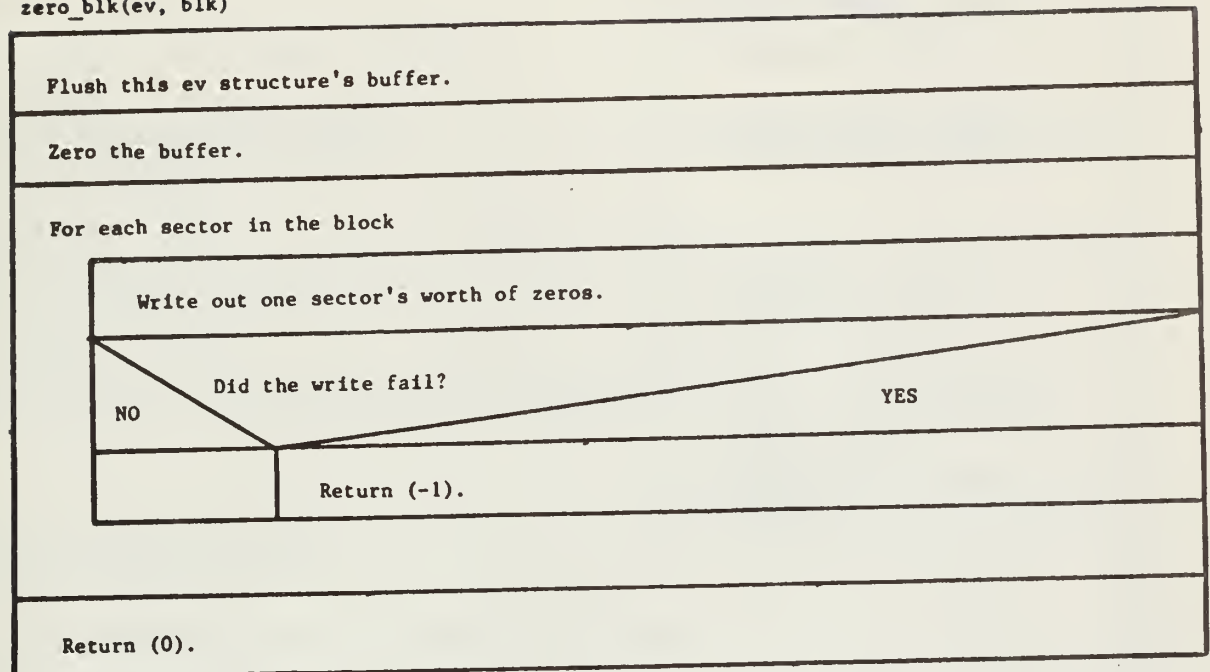
`xopen(fib, index_b, o, slot)`



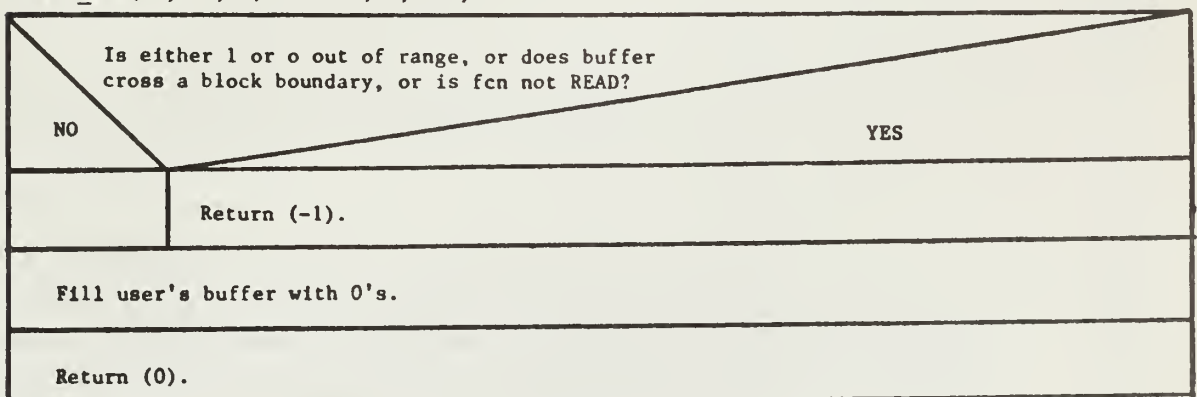
Z80_int



zero_blk(ev, blk)



zero_sim(d1, d2, o, userbuf, 1, fcn)



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

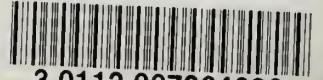
REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS
BEFORE COMPLETING FORM

1. REPORT NUMBER CAC Document Number 252 CCTC-WAD Document Number 7523		2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Intelligent Terminal Software Flowcharts		5. TYPE OF REPORT & PERIOD COVERED Research	
		6. PERFORMING ORG. REPORT NUMBER CAC #252	
7. AUTHOR(s) Deborah S. Brown David A. Willcox Betty Kasprzycki John R. Mullen		8. CONTRACT OR GRANT NUMBER(s) DCA100-76-C-0088	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Center for Advanced Computation University of Illinois Urbana, Illinois 61801		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS	
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center 11440 Isaac Newton Square, North Reston, Virginia 22090		12. REPORT DATE October 31, 1977	
		13. NUMBER OF PAGES	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report) Copies may be obtained from the address in 11 above.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) No restriction on distribution.			
18. SUPPLEMENTARY NOTES None			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Intelligent Terminal Man-Machine Interface			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This document contains Nassi-Shneiderman flowcharts for the Intelligent Terminal system described in CAC document number 236 (CCTC-WAD Document Number 7516), "Intelligent Terminal Programmer's Manual."			

UNIVERSITY OF ILLINOIS-URBANA

510.841L63C C001
CAC DOCUMENT\$URBANA
251-252 1977



3 0112 007264093